Областное государственное бюджетное профессиональное образовательное учреждение Южский технологический колледж

РАССМОТРЕН
методической комиссией № 2
Протокол № 1
от « 31 » 08 2023 г.
Председатель
_____ С.В.Вежновцева

УТВЕРЖДАЮ:
Зам.директора по УПР
_____ Е.И.Акопян

КОМПЛЕКТ ПРАКТИЧЕСКИХ ЗАНЯТИЙ
по учебной дисциплине
МДК 01.01 Технология выполнения оформления и компоновки технической документации
для обучающихся по специальности 09.01.03
Оператор информационных систем и ресурсов

Разработчик
А.Э Работин,
преподаватель

Южа – 2023г.

Практические занятия по общеобразовательному учебному предмету **МДК 01.01 Технология выполнения оформления и компоновки технической документации**

разработаны в соответствии с Федеральным государственным образовательным стандартом (далее–ФГОС) по профессии 09.01.03 «Оператор информационных систем и ресурсов» и предназначены для студентов очной формы обучения.

Объем практических работ по дисциплине **МДК 01.01 Технология выполнения оформления и компоновки технической документации** 90 часов.

Перечень практических работ соответствует содержанию программы **МДК 01.01 Технология выполнения оформления и компоновки технической документации**

2.1. Цель практических занятий: формирование у обучающихся базовой системы знаний и умений в области оформления и компоновки технической документации. Это направлено на освоение основного вида деятельности «Оформление и компоновка технической документации» и соответствующих ему профессиональных компетенций.

2.2. Задачи практических занятий:

- **Освоение инструментов создания текстовых документов**. Например, изучение основных текстовых редакторов, инструментов разметки, рассылки, рецензирования, колонтитулов.
- **Формирование навыков форматирования** текста в соответствии с условиями форматирования, разметки и соблюдением стиля.
- **Освоение внедрения в документы таблиц и иллюстраций**. Например, изучение форматирования таблиц и табличных данных, подготовки иллюстраций для вставки в документы.
- **Изучение преобразования и перекомпоновки документов**. Например, освоение сканирования текстовых документов, инструментов распознавания текста, применения импортирования и внедрения текстовых, табличных и графических объектов из разных программных приложений.
- **Освоение сохранения, копирования и создания резервных копий документов**.

При выполнении каждой практической работы обучающиеся знакомятся с целями и задачами работы.

Работа состоит из двух частей: теоретической (для ознакомления и опоры при выполнении задания) и практической, которая содержит несколько заданий (письменные ответы на вопросы.

Обязательным требованием к выполнению практической работы является вывод, который выполняется письменно.

В конце каждой практической работы содержатся критерии оценки, с которыми обучающиеся так же знакомятся.

Перечень практических работ

№ зан-я	Тема практического занятия	Кол-во часов
1.	Практическое занятие 1-2. Создание, настройка, применение стилей в документе с помощью текстового процессора	2
2.	Практическое занятие 3-4 . Создание информационно-поискового аппарата документа с помощью текстового процессора	2
3.	Практическое занятие 5-6. Нахождение в Интернете источников информации по заданной теме	2
4.	Практическое занятие 7-8. Использование ресурсов научно-технических библиотек и архивов	2
5.	Практическое занятие 9-10. Использование программ оптического распознавания символов	2
6.	Практическое занятие 11-12 . Использование русскоязычных и англоязычных источников научно - технического характера	2
7.	Практическое занятие 13- 14. Составление научно- технического текста, придерживаясь композиционных и стилистических правил, присущих научно-техническому стилю	2
8.	Практическое занятие 15 - 16 Настройка программы - редактор для работы с документами, размеченными в соответствии с правилами заданного приложения языкаXML	2
9.	Практическое занятие 17 - 18. Выбор и применение программы, предназначенные для преобразования документов, представленных в различных форматах, в документы, размеченные в соответствии с правилами языка разметки HTML или заданного приложения языка XML	2
10.	Практическое занятие 19 - 20. Составление несложных стилей на языке CascadeStyleSheets(CSS) и отлаживание их	2
11.	Практическое занятие 21-22. Преобразование рисунков (в том числе представленные в векторных форматах) в форматы PNG и JPEG.	2
12.	Практическое занятие 23 - 24. Обнаружение и устранение ошибок разметки в документе, который должен соответствовать правилам заданного языка разметки	2
13.	Практическое занятие 25 - 26 . Придача разметке документа удобно-читаемый структурированный вид	2
14.	Практическое занятие 27 - 28. Создание информативных графических схем	2
15.	Практическое занятие 29 - 30. Подготовка графических схем с помощью инструментов, предусмотренных в наиболее распространенных пакетах офисных приложений	2
16.	Практическое занятие 31 - 34. Подготовка графических схем с помощью специализированных визуальных редакторов	4
17.	Практическое занятие 35 - 38. Подготовка графических схем с помощью программ-генераторов, поддерживающих языки разметки, предназначенные для описания графических схем	4
18.	Практическое занятие 39 - 42. Преобразование файлов графических схем из векторных форматов в растровые форматы	4
19.	Практическое занятие 43 - 46. Отображение алгоритма или процесса с помощью блок-схемы, логическую структуру базы данных с помощью диаграммы "сущность - связь"	4
20.	Практическое занятие 47 - 50. Отображение структуры компьютерной сети или системы с помощью графической схемы	4
21.	Практическое занятие 51 - 54. Получение иллюстративные снимки экрана.	4
22.	Практическое занятие 55 - 58. Оптимизация снимка экрана для размещения на странице заданного формата	4
23.	Практическое занятие 59 - 62. Оптимизация цветности снимка экрана для воспроизведения заданным способом. Нанесение на снимок экрана поясняющие надписи и графические элементы	4

24.	Практическое занятие 63 - 66. Опрос экспертов и анализ полученных сведений	4
25.	Практическое занятие 67 - 70. Исследование программных средства на тестовом стенде	4
26.	Практическое занятие 71 - 74. Разработка плана документа и согласование его с экспертами	4
27.	Практическое занятие 75 - 78. Составление информационных текстовых файлов	4
28.	Практическое занятие 79 - 82. Разработка руководства пользователя несложного программного средства или краткого руководства по эксплуатации несложного технического средства	4
29.	Практическое занятие 83 - 86. Анализ замечаний экспертов и внесение исправлений в документ	4
30.	Практическое занятие 87 - 90. Преобразование документов в различные выходные форматы	4

Практическое занятие 1-2: Создание, настройка и применение стилей в документе

Цель занятия: Научиться профессионально форматировать документы с помощью стилей, что позволит ускорить работу, обеспечить единообразие оформления и легкость последующего редактирования.

Программное обеспечение: Microsoft Word (версии 2016, 2019, 2021 или Microsoft 365). Принципы аналогичны для других процессоров (LibreOffice Writer, Google Docs).

Необходимые знания: Базовые навыки работы с Word (ввод текста, выделение, изменение шрифта, размера).

План занятия и Ход работы

Часть 1: Теоретическая основа и знакомство со стилями (≈30 минут)

Задача: Понять, что такое стили и зачем они нужны.

1. Введение:

- **Что такое стиль?** Стиль — это заранее заданный набор параметров форматирования (шрифт, размер, выравнивание, интервалы и т.д.), который можно применить к тексту.
- **Зачем использовать стили?**
  - **Единообразие:** Все заголовки одного уровня, основные абзацы, списки и подписи выглядят одинаково по всему документу.
  - **Скорость:** Изменение оформления всего документа за секунды. Например, чтобы изменить шрифт всех заголовков 2 уровня, достаточно изменить один стиль.
  - **Автоматизация:** Автоматическое создание оглавления, навигационной области и структуры документа.
  - **Профессионализм:** Использование стилей — признак грамотной работы с документами.

2. Обзор панели стилей:

- Показать вкладку **«Главная»** → группа **«Стили»**.
- Объяснить основные встроенные стили: Заголовок 1, Заголовок 2, Название, Подзаголовок, Обычный, Основной абзац.

Часть 2: Практическая работа. Создание простого документа (≈1 час 15 минут)

Задача: Создать документ и применить к нему встроенные стили.

1. Подготовка текста:

- Создайте новый документ.
- Введите следующий текст:

Великие изобретения человечества

Введение

На протяжении всей истории человечество создавало удивительные вещи, которые коренным образом меняли нашу жизнь. В этом докладе мы рассмотрим некоторые из них.

Колесо

История колеса

Колесо считается одним из величайших изобретений. Оно появилось в эпоху неолита и первоначально использовалось в гончарном круге.

Влияние на транспорт

С изобретением колеса началась настоящая революция в транспорте. Появились повозки, телеги, а впоследствии и автомобили.

Печатный станок

Изобретение Иоганна Гутенберга в XV веке позволило наладить массовое производство книг. Это привело к распространению знаний и грамотности.

Заключение

Все эти изобретения объединяет одно: они дали человечеству новый инструмент для развития и преодоления препятствий.

2. Применение встроенных стилей:

- Выделите строку «**Великие изобретения человечества**» и примените стиль **«Название»**.
- Выделите «**Введение**» и примените стиль **«Заголовок 1»**.
- Выделите «**Колесо**» и «**Печатный станок**» и примените к ним стиль **«Заголовок 1»**.
- Выделите «**История колеса**» и «**Влияние на транспорт**» и примените к ним стиль **«Заголовок 2»**.
- Аналогично оформите «**Заключение**» как Заголовок 1.
- Ко всем основным абзацам текста (не-заголовкам) примените стиль **«Основной абзац»**.

3. Наблюдение за изменениями:

- Обратите внимание, как изменился внешний вид документа. Он стал структурированным.

4. Создание оглавления (демонстрация мощности стилей):

- Поставьте курсор в начало документа.
- Перейдите на вкладку **«Ссылки»** → **«Оглавление»** → выберите **«Автособираемое оглавление»**.
- Убедитесь, что оглавление создалось автоматически, основываясь на примененных стилях Заголовок 1 и Заголовок 2.

Часть 3: Настройка и создание собственных стилей (≈1 час 15 минут)

Задача: Научиться изменять существующие стили и создавать новые, соответствующие конкретным требованиям.

1. Изменение встроенного стиля:

- Предположим, нам не нравится синий цвет Заголовка 2. Давайте его изменим.
- На панели стилей найдите **«Заголовок 2»**. Щелкните по нему правой кнопкой мыши и выберите **«Изменить...»**.
- В открывшемся диалоговом окне:
  - Измените цвет шрифта на темно-зеленый.
  - Добавьте **полужирное** начертание.
  - Нажмите на кнопку **«Формат»** в левом нижнем углу и выберите **«Абзац»**. Установите интервал *После:* 6 pt.
  - Нажмите **«ОК»**.
- **Наблюдение:** Все заголовки второго уровня в документе мгновенно изменили свой вид.

2. Создание нового стиля (например, для цитат):

- Выделите любой абзац основного текста.
- На панели «Стили» нажмите на маленькую стрелочку в правом нижнем углу, чтобы открыть область стилей.
- Внизу области стилей нажмите кнопку **«Создать стиль»** (первая слева).

- Дайте стилю имя: Моя Цитата.
- Нажмите **«Изменить...»**.
- Настройте параметры:
  - *Основан на стиле:* «Основной абзац».
  - *Шрифт:* Курсив (Italic).
  - *Цвет:* Серый.
  - Нажмите **«Формат»** → **«Абзац»**:
    - *Выравнивание:* По ширине.
    - *Отступы: Слева:* 1,5 см, *Справа:* 1,5 см.
    - *Интервал: До:* 6 pt, *После:* 6 pt.
    - *Отступ первой строки:* Нет.
- Нажмите **«ОК»**, затем еще раз **«ОК»**.
- **Применение:** Найдите в тексте место для цитаты, введите ее и примените только что созданный стиль Моя Цитата.

Часть 4: Применение стилей к спискам и завершение работы (≈45 минут)
Задача: Научиться работать со стилями списков и закрепить материал.
1. Создание многоуровневого списка со стилями:
- В конце документа, после «Заключение», создайте новый раздел с заголовком «Список литературы» (Заголовок 1).
- Добавьте несколько пунктов списка литературы.
- Выделите их и на вкладке «Главная» нажмите **«Многоуровневый список»**.
- Выберите стиль списка, который включает номера страниц (например, для ГОСТа).
- Настройте стиль списка по своему усмотрению

2. Проверка и итоговое редактирование:
- Используйте область **«Навигация»** (вкладка «Вид» → галочка «Навигационная область»). Убедитесь, что вы видите четкую структуру своего документа. Это прямое следствие использования стилей заголовков.
- Внесите изменения в стиль Заголовок 1. Убедитесь, что изменения применились ко всем разделам и автоматически обновилось оглавление (для обновления оглавления щелкните по нему и нажмите «Обновить таблицу»).

3. Итоговое задание для самостоятельного выполнения (оставшееся время / дома):
- Создайте документ на свободную тему
- Требования:

1. Объем не менее 3-х разделов.
2. Использовать как минимум 2 уровня заголовков (настроив их внешний вид).
3. Создать и применить собственный стиль для основного текста.
4. Добавить блок с цитатой, оформив ее своим стилем.
5. Создать автоматическое оглавление в начале документа.
6. Предоставить итоговый файл.

Практическое занятие 3-4: Создание информационно-поискового аппарата документа

Цель занятия: Научиться создавать в документе динамические элементы, обеспечивающие навигацию, поиск информации и ссылочный аппарат: оглавление, указатели, перекрёстные ссылки, сноски и концеnotes.
Программное обеспечение: Microsoft Word (2016/2019/2021/Microsoft 365). Принципы аналогичны для LibreOffice Writer и Google Docs.
Необходимые знания: Уверенное использование стилей (результат занятия 1-2).

План занятия и Ход работы

Часть 1: Подготовка документа-основы (≈30 минут)
Задача: Создать или использовать готовый структурированный документ для работы.

1. **Вариант А (Продолжение):** Откройте документ, созданный на занятии 1-2 ("Великие изобретения человечества"). Дополните его новым разделом:
   Компьютер
   История развития
   Первые компьютеры, такие как **ENIAC**, занимали целые комнаты. Ключевой фигурой в истории вычислительной техники считается **Алан Тьюринг**.
   Современность
   Сегодня компьютеры интегрированы во все сферы жизни. Развитие **искусственного интеллекта** открывает новые горизонты.
2. **Вариант Б (Новый документ):** Создайте документ "Теория решения изобретательских задач (ТРИЗ)" со следующей структурой, оформленной стилями:
   - Заголовок 1: "Введение в ТРИЗ"
   - Заголовок 2: "Основные понятия" (в этом разделе упомяните "Идеальная конечная система (ИКР)", "Противоречия")
   - Заголовок 1: "Инструменты ТРИЗ"
   - Заголовок 2: "Алгоритм решения изобретательских задач (АРИЗ)"
   - Заголовок 2: "Вепол-анализ"
   - Заголовок 1: "Заключение"

   Важно: Убедитесь, что все заголовки оформлены стилями Заголовок 1, Заголовок 2 и т.д.

Часть 2: Углублённая работа с оглавлением (≈1 час)
Задача: Научиться настраивать и форматировать автоматическое оглавление.

1. **Вставка настраиваемого оглавления:**
   - Удалите старое оглавление, если оно было.
   - Перейдите **«Ссылки»** → **«Оглавление»** → **«Настраиваемое оглавление...»**.
   - В открывшемся диалоговом окне:
     - На вкладке **«Оглавление»** снимите галочку **«Показать номера страниц»**. Убедитесь, что в предпросмотре оглавление исчезло.
     - Поставьте галочку обратно.
     - Измените **«Заполнитель»** с точек на тире.

- Нажмите кнопку **«Параметры...»**. Убедитесь, что уровни оглавления связаны со стилями Заголовок 1 (уровень 1), Заголовок 2 (уровень 2) и т.д. Это основа автоматизации.
- Нажмите **«Изменить...»**. Вы увидите, что можно отдельно настроить стиль для каждого уровня оглавления (например, Оглавление 1, Оглавление 2). Измените стиль Оглавление 2 на курсив. Нажмите **ОК**.

2. **Обновление оглавления:**
- Внесите изменение в документ: добавьте новый заголовок где-нибудь в середине текста, оформив его стилем.
- Щелкните по оглавлению и нажмите кнопку **«Обновить таблицу»** (или клавиша **F9**).
- Выберите опцию **«Обновить целиком»**. Убедитесь, что новый раздел появился в оглавлении.

Часть 3: Сноски и концевые сноски (примечания) (≈45 минут)
Задача: Научиться добавлять пояснения и ссылки на источники.

1. **Добавление обычной сноски (пояснение внизу страницы):**
- Найдите в тексте упоминание "**ENIAC**" (или "**ИКР**" в документе про ТРИЗ).
- Поставьте курсор после этого слова.
- Перейдите **«Ссылки» → «Вставить сноску»** (или Ctrl+Alt+F).
- Word автоматически перенесет вас в низ страницы. Введите текст: "ENIAC (Electronic Numerical Integrator and Computer) — первый электронный цифровой компьютер общего назначения."
- Обратите внимание, что номер сноски является ссылкой. Щелчок по нему позволяет переходить между текстом и примечанием.

2. **Добавление концевой сноски (источники в конце документа):**
- Найдите предложение, где упоминается "**Алан Тьюринг**" (или "**АРИЗ**").
- Поставьте курсор после этого предложения.
- Перейдите **«Ссылки» → «Вставить концевую сноску»** (или Ctrl+Alt+D).
- Word перенесет вас в конец документа (перед оглавлением). Введите текст: "Эндрю Ходжес. Алан Тьюринг: Энигма."

3. **Настройка формата сносок:**
- Нажмите на маленькую стрелочку в группе «Сноски» на ленте.
- В диалоговом окне можно изменить формат номеров (арабские цифры, римские, символы), начать нумерацию заново и т.д. Например, для концевых сносок можно выбрать формат "[1]", "[2]".

Часть 4: Указатель (предметный индекс) (≈1 час)
Задача: Создать алфавитный указатель ключевых терминов.

1. **Разметка элементов указателя:**
- Найдите в тексте ключевой термин, например, **"искусственный интеллект"** (или "**противоречия**").
- Выделите это слово.
- Перейдите **«Ссылки» → «Пометить элемент» → «Выделить все»**. В открывшемся диалоговом окне Word покажет все вхождения этого термина. Нажмите **«Пометить все»**. Закройте окно.
- Проделайте то же самое для других терминов: **"компьютер"**, **"Тьюринг"** (или "**вепол-анализ**", "**ИКР**").

2. **Создание указателя:**
- Поставьте курсор в конец документа (после всех разделов, но до списка концевых сносок, если он есть).
- Напишите "**Указатель**" и оформите как Заголовок 1.
- Перейдите **«Ссылки» → «Оглавление» → «Указатель»**.
- В диалоговом окне вы можете выбрать формат (например, «Затейливый»), тип заполнителя, число колонок. Нажмите **ОК**.
- Указатель будет создан автоматически с номерами страниц, на которых встречаются помеченные термины.

3. **Обновление указателя:**
- Если вы добавили новые пометки, щелкните правой кнопкой по указателю и выберите **«Обновить поле»**.

Часть 5: Перекрёстные ссылки (≈30 минут)
Задача: Научиться создавать "гиперссылки" внутри документа.

1. **Создание ссылки на заголовок:**
- В разделе "Заключение" напишите новый абзац: "Как уже говорилось в разделе «», основные изобретения..."
- Поставьте курсор между кавычками.
- Перейдите **«Ссылки» → «Перекрёстная ссылка»**.
- В диалоговом окне:
  - **Тип ссылки:** Выберите "Заголовок".
  - **Вставить ссылку на:** Выберите "Текст заголовка".
  - В списке ниже выберите заголовок "Введение".
- Нажмите **«Вставить»**. Слово "Введение" станет ссылкой, по которой можно перейти, удерживая Ctrl.

2. **Создание ссылки на номер рисунка:**
- (Если есть время) Вставьте в документ любое изображение (например, из Интернета). Подпишите его, используя **«Ссылки» → «Вставить название»** (тип "Рисунок", подпись "Схема ENIAC").
- В тексте напишите: "(см. рис.)". Используя «Перекрёстная ссылка» (Тип: "Рисунок", Вставить: "Только номер и подпись"), вставьте ссылку на этот рисунок.

Итоговое задание для самостоятельного выполнения:
Создать сложноструктурированный документ (например, "Обзор современных языков программирования") со следующими элементами ИПА:

1. **Структура:** Не менее 3 разделов и 2 уровней заголовков (оформлены стилями).
2. **Оглавление:** Настроенное оглавление с нестандартным заполнителем.
3. **Сноски:** Не менее 2-х обычных и 2-х концевых сносок
4. **Указатель:** Указатель, содержащий не менее 5 ключевых терминов.
5. **Перекрёстные ссылки:** Не менее 2-х перекрёстных ссылок

Критерии успешного выполнения:
- Все элементы ИПА созданы автоматическими средствами Word, а не вручную.

- При изменении текста, добавлении страниц или заголовков оглавление, указатель и перекрёстные ссылки корректно обновляются.
- Документ представляет собой целостную, удобную для навигации систему.

Практическое занятие 5-6: Нахождение в Интернете источников информации по заданной теме

Цель занятия: Научиться проводить эффективный информационный поиск, критически оценивать найденные источники и систематизировать результаты для их дальнейшего использования в учебной и профессиональной деятельности.

Программное обеспечение: Компьютер с доступом в интернет и браузером (Chrome, Firefox, Edge и т.д.).

Необходимые знания: Базовые навыки работы с браузером.

План занятия и Ход работы

Часть 1: Теоретический фундамент (≈ 45 минут)

Задача: Понять принципы работы поисковых систем и освоить технику эффективного поиска.

1. Введение: Проблема "шума", а не "дефицита" информации.

- Обсуждение: Почему сегодня сложно найти *качественную* информацию? (Много рекламы, SEO-оптимизированный, но бесполезный контент, фейки, мнения вместо фактов).

2. Как работают поисковые системы (ПС) — упрощенно:

- **Робот (краулер):** Обходит сайты и собирает информацию.
- **Индекс:** Гигантская база данных просканированных страниц.
- **Алгоритм:** Оценивает релевантность страницы вашему запросу и ранжирует результаты.

3. Язык поисковых запросов: Искусство задавать правильные вопросы.

- **Ключевые слова:** Основные понятия темы. Не "про животных", а "среда обитания бурого медведя".
- **Использование кавычек** для поиска точной фразы.
  - *Пример:* искусственный интеллект (десятки миллионов результатов) vs. "искусственный интеллект" (точное словосочетание).
- **Оператор** site: для поиска на конкретном сайте.
  - *Пример:* site:gov экологическая безопасность (поиск официальной информации на государственных сайтах).
- **Оператор - (минус)** для исключения слов.
  - *Пример:* Пирамида Хеопса -казино -отель (исключает коммерческие объекты).
- **Использование * (звездочка)** как замены неизвестного слова.
  - *Пример:* "первым изобрел *".
- **Поиск по типам файлов:**
  - *Пример:* filetype:pdf теория решения изобретательских задач.

Часть 2: Практикум по эффективному поиску (≈1 час 15 минут)

Задача: Закрепить технику составления поисковых запросов на практике.

Упражнение 1: От общего к частному.

- **Тема:** "Влияние социальных сетей на психическое здоровье подростков".
- **Задание:** Составьте цепочку запросов от самого общего к самому конкретному.

1. *Общий запрос:* социальные сети подростки
2. *Уточнение:* влияние социальных сетей на подростков
3. *Конкретизация:* "тревожность" "социальные сети" подростки исследование
4. *Поиск в академическом источнике:* site:cyberleninka.ru влияние Instagram на самооценку подростков

Упражнение 2: Поиск официальной и статистической информации.

- **Тема:** "Демографическая ситуация в России".
- **Задание:** Найдите официальную статистику.

1. Используйте оператор site: для поиска на сайтах: site:rosstat.gov.ru, site:gks.ru.
2. Найдите последний "Демографический ежегодник России".
3. *Запрос:* site:gks.ru "естественная убыль населения" 2023 filetype:pdf

Упражнение 3: Поиск специализированной информации.

- **Тема:** "Использование 3D-печати в строительстве".
- **Задание:** Найдите патент, научную статью и новостной репортаж.

1. *Патент:* site:rupto.ru 3D-печать строительство
2. *Научная статья:* site:elibrary.ru "3D-печать" "строительство" 2022
3. *Новость:* "напечатали дом на 3D-принтере"

Часть 3: Критическая оценка источников (≈45 минут)

Задача: Научиться отличать надежные источники от ненадежных.

Памятка "Проверка источника (CRAAP Test - упрощенная версия)":

- **Currency (Актуальность):** Когда была опубликована информация? Она все еще актуальна? Для науки — 3-5 лет, для IT — 1-2 года.
- **Relevance (Релевантность):** Отвечает ли источник на ваш вопрос? Для какой аудитории он предназначен (для школьников или для ученых)?
- **Authority (Авторитетность):** Кто автор? Какая у него квалификация? Кто опубликовал информацию (вуз, государственное учреждение, блогер)?
- **Accuracy (Достоверность):** Подтверждается ли информация доказательствами (ссылки на исследования, статистику)? Можно ли ее проверить в других источниках?
- **Purpose (Цель):** Какова цель информации? Проинформировать, убедить, продать что-то? Нет ли скрытой рекламы или предвзятости?

Практическое задание "Следопыт":

- Преподаватель дает список из 3-4 ссылок по одной теме (например, "польза и вред ГМО").
- Список включает: научный обзор с elibrary.ru, статью с популярного медицинского портала, статью с сайта "про здоровый образ жизни" с продажей БАДов, новость с сайта-агрегатора.
- **Задание для студентов:** Проанализировать каждый источник по критериям CRAAP и расположить их в порядке убывания достоверности. Обосновать свой выбор.

Часть 4: Систематизация и сохранение результатов (≈45 минут)

Задача: Научиться организовывать найденную информацию для последующей работы.

1. Сохранение веб-страниц:

- **Закладки браузера:** Создание тематических папок.
- **Сервисы для сохранения и аннотирования:** Pocket, Notion, Evernote. (Показать на примере одного из них).

2. Создание аннотированной библиографии (практическое задание):

- **Цель:** Не просто скопировать ссылку, а кратко зафиксировать суть источника.

- **Формат (создать таблицу в Word или Google Docs):**

№	Название источника, ссылка	Автор, источник публикации	Краткое описание (2-3 предложения: о чем, основные выводы)	Оценка достоверности (Высокая/Средняя/Низкая)
1	Влияние социальных сетей на успеваемость студентов	Иванов А.И., журнал "Психология образования"	Исследование показало корреляцию между временем в соцсетях и средним баллом. Выборка: 500 человек.	Высокая (научная статья, рецензируемый журнал)
2	Почему TikTok крадет ваше время	Блог "Цифровая грамотность"	Автор блогер-психолог приводит 5 приемов, которые используют соцсети для удержания внимания.	Средняя (экспертное мнение, но без ссылок на исследования)

3. Правильное цитирование веб-источников:

- Краткий обзор форматов (ГОСТ, APA). Показать на примере, как оформить ссылку на сайт.
- *Пример (ГОСТ):* Влияние социальных сетей на когнитивные способности [Электронный ресурс] // Научный портал. URL: https://example.com (дата обращения: 25.10.2023).

Итоговое задание для самостоятельного выполнения:

Тема: Выбрать одну из предложенных тем (например, "Развитие зеленой энергетики в стране", "Этика искусственного интеллекта", "История и перспективы космического туризма").

Задание:

1. Используя изученные техники, найти **5-7 источников** информации по выбранной теме.
2. Источники должны быть **разных типов**: 1-2 научные статьи, 1 официальный отчет (гос.органа или международной организации), 1-2 качественные аналитические статьи, 1 новостной репортаж.
3. Создать документ, содержащий:
   - Краткое описание вашей стратегии поиска (какие ключевые слова и операторы вы использовали).
   - Аннотированную библиографию в виде таблицы.
   - Вывод: какой из источников вы считаете наиболее ценным для глубокого изучения темы и почему.

Критерии успешного выполнения:

- Демонстрация владения продвинутыми приемами поиска.
- Разнообразие и соответствие типов найденных источников.
- Глубина и критичность анализа в аннотированной библиографии.
- Грамотное оформление результатов.

Практическое занятие 7-8: Использование ресурсов научно-технических библиотек и архивов

Цель занятия: Сформировать практические навыки поиска, заказа и работы с профильными информационными ресурсами в научно-технических библиотеках (НТБ) и архивах. Науться отличать их возможности от общедоступного интернета.

Место проведения: Компьютерный класс с доступом в интернет *и/или* посещение реальной НТБ/архива (если возможно).

Необходимые знания: Навыки из занятия 5-6 (поиск в интернете).

План занятия и Ход работы

Часть 1: Введение в мир "академической" информации (≈45 минут)

Задача: Понять миссию и уникальность НТБ и архивов.

1. Чем НТБ и архивы лучше интернета?

- **Концентрация проверенных знаний:** Отсутствие "информационного шума".
- **Уникальные и редкие издания:** Диссертации, отчеты по НИР, депонированные рукописи, патенты, стандарты, специализированные периодические издания.
- **Глубина ретроспективы:** Журналы и книги 50-100-летней давности, часто не оцифрованные.

- **Научный аппарат:** Система каталогов и указателей, созданная профессионалами.

2. Типы информационных ресурсов:

- **Первичные источники (архивы):** Отчеты, чертежи, лабораторные журналы, приказы, личная переписка ученых и инженеров. *Это сырые данные, "следы" истории.*
- **Вторичные источники (НТБ):** Монографии, учебники, научные статьи, обзоры. *Это обработанная и проанализированная информация.*

3. Структура НТБ:

- Читальные залы (общие, отраслевые).
- Абонемент.
- Справочно-библиографический аппарат (СБА): каталоги, картотеки.
- Отдел редких книг и рукописей.
- Электронные читальные залы.

Часть 2: Практикум по работе с электронными каталогами НТБ (≈1 час 30 минут)

Задача: Освоить основной инструмент поиска в современной библиотеке.

1. Знакомство с крупнейшими НТБ России (виртуальный тур):

- **ГПНТБ России (Государственная Public Scientific and Technical Library):** www.gpntb.ru. Крупнейшая в стране.
- **РГБ (Российская государственная библиотека, "Ленинка"):** www.rsl.ru.
- **ВГБИЛ (Всероссийская государственная библиотека иностранной литературы):** www.libfl.ru.
- **Библиотеки при ведущих вузах:** НТБ МГУ, НТБ МИФИ, НТБ СПбПУ и т.д.

2. Упражнение 1: Поиск по электронному каталогу ГПНТБ.

- **Задача:** Найти монографию по теме "История развития космонавтики в СССР".
- **Ход работы:**

1. Открываем сайт ГПНТБ → раздел "Электронный каталог".
2. **Простой поиск:** Вводим ключевые слова: космонавтика СССР история.
3. **Анализ результатов:** Обращаем внимание на поля: Автор, Заглавие, Год издания, Место хранения (шифр).
4. **Расширенный поиск:** Используем фильтры:
   - *Вид документа:* Книга.
   - *Год издания:* с 1960 по 1990.
   - *Язык:* Русский.
5. Найдя нужную книгу, изучаем **электронный каталог** – какие еще тематические рубрики ей присвоены? (например, "Космонавтика -- История -- СССР").

3. Упражнение 2: Поиск периодики.

- **Задача:** Найти журнал "Доклады Академии наук" за последние 3 года.
- **Ход работы:**

1. В каталоге ищем "Доклады Академии наук".
2. Определяем, есть ли он в электронном виде (ссылка на полный текст) или только в бумажном.
3. Учимся читать **"ведомость наличия"** – таблицу, показывающую, какие номера за какие годы есть в фонде.

4. Упражнение 3: Поиск специализированных документов.

- **Задача:** Найти диссертацию по теме "разработка композитных материалов".
  - Используем сайт **РГБ** (так как она является основным хранителем диссертаций).
  - Ищем в **"Электронной библиотеке диссертаций"** РГБ. Обсуждаем, что полные тексты доступны только из читальных залов РГБ и ее партнеров.

Часть 3: Работа с архивными ресурсами (≈1 час)

Задача: Понять логику организации архивов и научиться делать первые шаги в поиске.

1. Основные понятия:

- **Фонд** – совокупность документов, образовавшихся в деятельности одного учреждения или лица.
- **Опись** – структурная единица фонда, перечень единиц хранения.
- **Единица хранения (ед. хр.)** – физически обособленный документ или группа документов (дело) в архиве.

2. Знакомство с порталами архивов:

- **ГАРФ (Государственный архив РФ):** statearchive.ru.
- **РГАЭ (Российский государственный архив экономики):** rgae.ru.
- **Портал "Архивы России":** rusarchives.ru – единая точка входа.

3. Практическое задание "Визитка архива":

- Студентам раздаются названия разных архивов (РГАЭ, РГАКФД (кинофото), ЦГАМО (Московской области) и т.д.).
- **Задание:** За 20 минут найти на сайте архива и записать в таблицу:

1. Основную тематику документов архива (что хранит?).
2. Примеры известных фондов или коллекций.
3. Условия доступа для исследователей (требуется ли письмо от вуза?).
4. Есть ли оцифрованные описи или документы?

4. Разбор кейса "Запрос в архив":

- Обсуждение, как правильно составить письменный запрос в архив.
- *Пример:* "Прошу предоставить для изучения в читальном зале опись № X фонда № Y "Главное управление по проектированию металлургических заводов" за 1950-1960 гг."

Часть 4: Комплексная задача и систематизация результатов (≈45 минут)

Задача: Применить все полученные навыки для решения конкретной исследовательской задачи.

Сквозное задание "На заре космической эры":

- **Исходная задача:** Подготовить список источников для исследования на тему "Роль С.П. Королева в создании первого искусственного спутника Земли".

План выполнения для студентов:

1. **Этап 1 (НТБ):** Найти в электронном каталоге РГБ или ГПНТБ:
   - Монографию С.П. Королева (или о нем), изданную при его жизни.
   - Сборник документов по истории космонавтики.
   - Научно-популярный журнал ("Техника - молодежи", "Наука и жизнь") за октябрь-ноябрь 1957 года.
2. **Этап 2 (Архив):** Найти на портале "Архивы России":

- Какой архив может хранить личный фонд С.П. Королева? (Подсказка: РГАНТД – Российский государственный архив научно-технической документации).
- Найти описание его фонда на сайте РГАНТД.

3. **Оформление результатов:** Создать сводную таблицу с колонками: "Тип источника", "Название/Автор", "Год", "Место хранения/Шифр/Ссылка", "Краткая аннотация".

Итоговое задание для самостоятельного выполнения:

Тема: Выбрать одну из предложенных тем, связанную с историей науки или техники в России/СССР (например, "Развитие вычислительной техники в СССР в 1960-е гг.", "Строительство БАМа: инженерный аспект", "Творчество авиаконструктора А.Н. Туполева").

Задание:

1. Составить библиографический список из **5-7 наименований**, используя ресурсы НТБ и архивов.
2. Список должен включать:
   - Не менее 2-3 книг или статей из фондов НТБ (найденных через электронные каталоги).
   - Не менее 1 единицы периодики (журнала или газеты) за период, близкий к событиям.
   - Описание 1-2 архивных фондов, непосредственно связанных с темой (с указанием номера фонда, описи и краткой характеристики документов).
3. Написать краткий отчет (0,5-1 стр.) о стратегии поиска: какие каталоги использовались, какие ключевые слова и фильтры применялись, с какими трудностями столкнулись.

Критерии успешного выполнения:

- Релевантность и разнообразие подобранных источников.
- Корректность использования библиотечной и архивной терминологии.
- Глубина поиска (использование расширенных возможностей каталогов, выход на архивные описи).
- Грамотное библиографическое описание источников.

Практическое занятие 9-10: Использование программ оптического распознавания символов (OCR)

Цель занятия: Освоить принципы работы и получить практические навыки использования программ оптического распознавания символов для преобразования отсканированных изображений и PDF-файлов в редактируемый текстовый формат.

Программное обеспечение:

- **Платное:** ABBYY FineReader (профессиональный стандарт).
- **Бесплатное:**
  - Онлайн-сервисы (Google Документы, OnlineOCR.net).
  - Встроенные средства (Microsoft Word, Adobe Acrobat Reader DC).
- **Сканер** или **камера смартфона** для получения изображений.

Необходимые знания: Базовые навыки работы с файлами, сканирования и обработки изображений.

План занятия и Ход работы

Часть 1: Теоретические основы OCR (≈30 минут)

Задача: Понять, как работает технология и где она применяется.

1. Что такое OCR (Optical Character Recognition)?

- **Определение:** Технология, которая преобразует изображения текста (отсканированные документы, фото) в машиночитаемые текстовые данные.
- **Аналогия:** Это "зрение" для компьютера, позволяющее ему "прочитать" текст с картинки.

2. Основные этапы распознавания:

- **Предобработка изображения:** Устранение перекоса, зашумленности, регулировка контрастности.
- **Сегментация:** Разделение изображения на блоки (текст, таблицы, картинки) и строки, а затем на отдельные символы.
- **Распознавание символов:** Сравнение выделенного контура символа с эталонными шаблонами (с использованием нейронных сетей).
- **Постобработка:** Проверка правописания, анализ контекста для исправления ошибок (например, "р" и "п").

3. Сферы применения OCR:

- Оцифровка архивов, книг, документов.
- Создание поисковых индексов для сканов.
- Автоматизация ввода данных (анкеты, бланки, счета).
- Распознавание текста с фото (объявления, визитки, ценники).
- Помощь людям с нарушениями зрения (озвучка текста).

Часть 2: Подготовка исходных материалов (≈30 минут)

Задача: Научиться правильно создавать "сырые" данные для распознавания.

Практическое задание "Создаем коллекцию для распознавания":

1. **Создание качественного скана:**
   - Возьмите чистый печатный текст (страницу из книги, распечатку).
   - Положите ровно в сканер. Отсканируйте с разрешением **300 dpi** в черно-белом или сером режиме. Сохраните как `scan_good.tiff` или `scan_good.pdf`.
   - *Обсуждение:* Почему 300 dpi? (Высокое качество, но не чрезмерный размер файла).
2. **Создание "проблемного" изображения:**
   - Создайте несколько "плохих" исходников:
     - **Перекос:** Сознательно положите лист неровно и отсканируйте (`scan_skew.jpg`).
     - **Тень/загиб:** Сфотографируйте страницу из толстой книги так, чтобы у корешка была тень (`photo_shadow.jpg`).
     - **Рукописный текст:** Напишите несколько предложений от руки разборчивым почерком (`handwritten.jpg`).
     - **Низкое качество:** Сфотографируйте текст на телефоне с большого расстояния или в плохом освещении (`photo_bad.jpg`).

Часть 3: Практикум с ABBYY FineReader (≈1 час 30 минут)

Задача: Освоить работу с профессиональным инструментом.

1. Знакомство с интерфейсом:

- Обзор панелей: Панель инструментов, область страниц, область текста, панель свойств.

2. Упражнение 1: Распознавание качественного скана.

- Откройте scan_good.tiff в FineReader.
- Запустите автоматическое распознавание.
- **Анализ результатов:**
  - Сравните исходное изображение и распознанный текст.
  - Оцените процент ошибок (он должен быть минимальным).
  - Сохраните результат в формате **DOCX** (сохраняет форматирование) и **TXT** (только чистый текст).

3. Упражнение 2: Работа с "проблемными" изображениями.

- Откройте scan_skew.jpg.
- **Вручную исправьте перекос:** Инструмент **«Исправить перекос страницы»**.
- Запустите распознавание. Сравните результат с тем, что было бы без исправления.
- Откройте photo_shadow.jpg.
- Используйте инструменты **«Улучшить изображение»** (яркость, контрастность) для устранения теней.
- Запустите распознавание.

4. Упражнение 3: Распознавание сложных макетов.

- Найдите или создайте PDF-файл, содержащий **текст в несколько колонок**, **таблицы** и **рисунки**.
- Откройте его в FineReader.
- **Проанализируйте, как программа автоматически определила блоки.**
- **Вручную проверьте и откорректируйте блоки:** Объедините ошибочно разделенные текстовые блоки, укажите тип блока (текст, таблица, изображение).
- Запустите распознавание.
- Сохраните результат в **DOCX**. Убедитесь, что колонки, таблицы и картинки сохранились корректно.

Часть 4: Альтернативные инструменты и онлайн-сервисы (≈45 минут)

Задача: Познакомиться с бесплатными и доступными методами распознавания.

1. Распознавание средствами Microsoft Word:

- Откройте Microsoft Word.
- **«Файл»** → **«Открыть»** → выберите ваш scan_good.pdf.
- Word покажет предупреждение, что будет выполнен процесс преобразования. Согласитесь.
- **Плюсы:** Простота, интеграция в знакомый интерфейс.
- **Минусы:** Меньшая точность и гибкость по сравнению с FineReader.

2. Использование Google Документов:

- Загрузите файл scan_good.jpg на Google Диск.
- Откройте его правой кнопкой мыши → **«Открыть с помощью»** → **«Google Документы»**.
- Система автоматически создаст новый документ, где вверху будет вставлено изображение, а ниже — распознанный текст.
- **Плюсы:** Полная бесплатность, доступность с любого устройства.
- **Минусы:** Требует интернет, меньше контроля над процессом.

3. Онлайн-сервисы (например, OnlineOCR.net):

- Зайдите на сайт онлайн-сервиса.
- Загрузите файл, выберите язык и формат выходного файла.
- Скачайте результат.
- **Обсуждение:** Плюсы (доступность) и минусы (конфиденциальность, ограничения на размер/количество файлов) онлайн-OCR.

Часть 5: Сравнительный анализ и постобработка (≈45 минут)

Задача: Научиться оценивать качество распознавания и исправлять ошибки.

Практическое задание "Контроль качества":

1. Возьмите один и тот же "проблемный" файл (например, photo_shadow.jpg).
2. Распознайте его тремя способами: в **FineReader**, через **Microsoft Word** и в **Google Документах**.
3. Создайте таблицу для сравнения:

Критерий	ABBYY FineReader	Microsoft Word	Google Документы
Точность текста			
Сохранение форматирования			
Сохранение таблиц/рисунков			
Удобство и скорость			
Вывод			

4. **Постобработка текста:**
   - Откройте распознанный текст в Word.
   - Включите отображение **непечатаемых символов** (¶).
   - Найдите и исправьте типичные ошибки OCR:
     - Замена похожих символов: «о» (латинская) на «о» (кириллическая), «rn» на «m».
     - «Мусорные» символы по краям.
     - Ошибочное определение абзацев.

Итоговое задание для самостоятельного выполнения:

Сценарий: "Вы получили отсканированную копию старой технической статьи (5-7 страниц) в формате PDF, содержащую текст в две колонки, формулы, таблицы и схемы".

Задание:

1. Найти или создать подобный многостраничный PDF-документ со сложной структурой.
2. Провести его полный цикл обработки в ABBYY FineReader (или аналоге):
   - Проверить и исправить перекосы.
   - Вручную проверить и откорректировать зоны распознавания на всех страницах.
   - Выполнить распознавание.
3. Сохранить результат в двух форматах: **DOCX** (для сохранения макета) и **TXT** (для использования в системах анализа текста).
4. Подготовить краткий отчет (1-2 страницы), который включает:
   - Описание возникших проблем при распознавании.
   - Методы их решения.
   - Оценку качества результата (в %) и перечень наиболее частых ошибок распознавания.
   - Выводы о целесообразности использования OCR для подобных документов.

Критерии успешного выполнения:

- Умение управлять процессом распознавания, а не просто нажимать кнопку.
- Качество итогового документа (минимальное количество ошибок, сохраненная структура).
- Глубина анализа и выводов в отчете.

Практическое занятие 11-12: Использование русско- и англоязычных источников научно-технического характера

Цель занятия: Научиться эффективно находить, критически оценивать, анализировать и корректно использовать в своей работе научно-техническую информацию на русском и английском языках. Сформировать понимание особенностей международной научной коммуникации.

Программное обеспечение: Компьютер с доступом в интернет, браузер, текстовый процессор (Word), системы машинного перевода (DeepL, Google Translate).

Необходимые знания: Навыки из предыдущих занятий (поиск в интернете, работа с библиотечными каталогами, использование стилей).

План занятия и Ход работы

Часть 1: Введение в мировое научное пространство (≈45 минут)

Задача: Понять роль английского языка как языка международной науки и место русскоязычных источников в нем.

1. Роль английского языка в науке ("Lingua Franca"):

- **Статистика:** >90% индексируемых научных журналов публикуются на английском.
- **Причины:** Глобализация, необходимость широкого цитирования, международные коллаборации.
- **Вывод:** Без доступа к англоязычным источникам невозможно быть в курсе современных мировых исследований.

2. Стратегия работы с источниками:

- **Русскоязычные источники:** Идеальны для быстрого погружения в тему, изучения фундаментальных основ, поиска информации по региональной/национальной проблематике.
- **Англоязычные источники:** Необходимы для выхода на передовой край науки, проведения актуального исследования, публикации в международных журналах.

3. Типы научно-технических источников (сравнительная таблица):

Тип источника	Русскоязычные примеры	Англоязычные примеры
Научные журналы	"Доклады Академии наук", "Известия РАН", "Прикладная механика"	Nature, Science, The Lancet, IEEE Transactions
Реферативные БД	eLibrary.ru, CyberLeninka	Scopus, Web of Science, PubMed, IEEE Xplore

Патенты	ФИПС (RU патенты)	USPTO (US), EPO (Europe), WIPO (международные)
Стандарты	ГОСТ, РСТ	ISO, IEC, IEEE
Препринты	arXiv.org (разделы на русском)	arXiv.org (основной поток на англ.)

Часть 2: Поиск и отбор источников (≈1 час 30 минут)
Задача: Освоить практические приемы поиска в ключевых русскоязычных и англоязычных базах данных.
Упражнение 1: Поиск в русскоязычных базах данных.

- **Тема для примера:** "Аддитивные технологии в авиастроении".
- **Платформа: eLibrary.ru** (Научная электронная библиотека).

1. **Простой поиск:** Ввод ключевых слов.
2. **Расширенный поиск:**
   - Использование фильтров: "Год публикации" (последние 5 лет), "Тип публикации" (статья в журнале), "Ядро РИНЦ".
   - Поиск по **списку ВАК** (журналы, рецензируемые Высшей аттестационной комиссией).
3. **Анализ статьи:** Найдя релевантную статью, проанализировать ее аннотацию, ключевые слова, список литературы (библиографию) – это источник новых идей для поиска!

Упражнение 2: Поиск в англоязычных базах данных.

- **Тема для примера:** "Additive manufacturing in aerospace engineering".
- **Платформа: Google Scholar** (свободно доступен) и/или **IEEE Xplore** (если доступен через вуз).

1. **Google Scholar:**
   - Поиск по точной фразе в кавычках: "additive manufacturing" "titanium alloy".
   - Использование фильтра "Создать alert" для отслеживания новых статей.
   - Анализ метрики "Цитируется в" – поиск более новых работ, которые ссылаются на эту.
2. **IEEE Xplore (или аналоги):**
   - Использование профессиональных фильтров: "By Subject", "Conference Location", "Publication Year".

Упражнение 3: Сравнительный анализ.

- **Задание:** По одной теме найти по 2 статьи в eLibrary и Google Scholar.
- Заполнить таблицу:

Критерий	Статья 1 (RUS)	Статья 2 (ENG)
Глубина раскрытия темы		
Новизна (год публикации)		
Наличие эксперимента/расчетов		
Качество библиографии		
Общее впечатление		

Часть 3: Чтение, анализ и перевод (≈1 час)
Задача: Научиться эффективно работать с англоязычным текстом, преодолевая языковой барьер.
1. Алгоритм работы со статьей:

1. **Аннотация (Abstract)** – понять, о чем статья и релевантна ли она.
2. **Введение (Introduction)** – понять контекст и проблему.
3. **Заключение (Conclusion)** – узнать основные выводы.
4. **Просмотр рисунков и графиков** – визуальное понимание результатов.
5. **Глубокое чтение** всего текста (если статья релевантна).

2. Работа с машинным переводом (DeepL, Google Translate):

- **"Слепое" доверие vs. Осознанное использование.**
- **Практикум:**

1. Возьмите абзац из англоязычной статьи.
2. Переведите его с помощью DeepL.
3. **Критически проверьте перевод:**
   - Технические термины: cell в биологии – "клетка", в технике – "элемент", "ячейка".
   - Сложные грамматические конструкции.
   - Используйте англо-русские **отраслевые словари** (например, Multitran).
4. Сравните с "сырым" переводом. Сделайте вывод о необходимости владения базовой научной лексикой.

3. Составление глоссария:

- **Задание:** При чтении 2-3 англоязычных статей по вашей теме выпишите 10-15 ключевых терминов с переводом и определением.
  - *Пример:* **Fatigue strength** – Предел выносливости – Максимальная нагрузка, которую материал может выдержать без разрушения после заданного числа циклов.

Часть 4: Интеграция источников и академическая добросовестность (≈45 минут)
Задача: Научиться корректно использовать идеи других авторов в своей работе.
1. Правила цитирования:

- **Прямое цитирование** (дословно) – используется редко, в кавычках с обязательной ссылкой.
- **Непрямое цитирование** (пересказ своими словами) – используется часто, но также требует ссылки на источник.
- **Ссылка на идею/метод** – указание на работу, где этот метод был впервые предложен.

2. Что такое плагиат?

- Обсуждение: Заимствование текста, идей, данных без указания автора.
- **Последствия:** Академические (отчисление), репутационные, юридические.

3. Практикум "Свои vs. Чужие слова":

- Дается короткий абзац из научного текста.
- **Задание:** Переписать этот абзац своими словами, сохранив смысл, и корректно оформить ссылку на оригинал.
- *Пример оригинала:* "Результаты показывают, что использование карбида кремния в качестве анодного материала позволяет увеличить емкость литий-ионного аккумулятора на 15%."
- *Пример корректного пересказа:* "Как установлено в исследовании [1], применение карбида кремния для создания анода способствует росту удельной емкости литий-ионных батарей приблизительно на 15%."

4. Обзор систем цитирования:

- **ГОСТ Р 7.0.5-2008** – основной стандарт в России.
- **APA (American Psychological Association), IEEE** – международные стандарты.

Итоговое задание для самостоятельного выполнения:

Тема: Актуальная научно-техническая проблема (напр., "Водородная энергетика", "Биосовместимые материалы", "Квантовые вычисления").

Задание:

1. **Поиск источников:** Подобрать библиографический список из **8-10 источников** для написания обзорной статьи по выбранной теме.
2. **Требования к списку:**
   - **4-5 источников** на английском языке (статьи из международных журналов, материалы конференций).
   - **3-4 источника** на русском языке (статьи из журналов ВАК, монографии).
   - Не менее **2 источников**, опубликованных за последние 2 года.
3. **Аналитическая записка:** На основе подобранных источников написать аналитическую записку (1-1.5 стр.), которая включает:
   - Краткое описание текущего состояния проблемы.
   - Сравнительный анализ подходов в российских и зарубежных исследованиях.
   - Вывод о перспективных направлениях развития.
4. **Оформление:** Оформить список литературы в соответствии с ГОСТом.

Критерии успешного выполнения:

- Релевантность, новизна и разнообразие подобранных источников.
- Глубина сравнительного анализа в записке.
- Демонстрация навыка осмысленного перевода и работы с терминологией.
- Абсолютная корректность цитирования и оформления списка литературы.

Практическое занятие 13-14: Составление научно-технического текста

Цель занятия: Освоить композиционные, стилистические и языковые нормы научно-технического стиля. Научиться создавать логически стройный, объективный и точный текст, соответствующий требованиям академического сообщества.

Программное обеспечение: Текстовый процессор (Microsoft Word), доступ к интернету для проверки источников.

Необходимые знания: Навыки работы со стилями, поиска и анализа источников, базовые навыки написания текстов.

План занятия и Ход работы

Часть 1: Введение в научно-технический стиль (НТС) (≈45 минут)

Задача: Понять цели, задачи и основные черты научного стиля.

1. Три кита НТС:

- **Точность** – однозначность понимания, исключение двусмысленностей.
- **Объективность** – беспристрастное изложение фактов, минимум авторской оценки.
- **Логичность** – последовательное, аргументированное изложение материала.

2. Языковые особенности НТС:

- **Лексика:**
  - Терминология (однозначные, общепринятые в научной среде понятия).
  - Отсутствие разговорной, эмоционально-окрашенной и устаревшей лексики.
  - Использование абстрактных существительных (анализ, синтез, исследование, явление).
- **Морфология:**
  - Преобладание существительных над глаголами.
  - Использование глаголов в настоящем времени («результаты *показывают*»).
  - Широкое использование причастий и деепричастий.
- **Синтаксис:**
  - Преобладание сложных предложений с четкой логической связью.
  - Использование вводных слов и конструкций (следовательно, таким образом, в результате).
  - Широкое использование пассивных (страдательных) конструкций («*был проведен эксперимент*» вместо «*мы провели эксперимент*»).

3. Композиция научного текста (классическая структура IMRaD):

- **I**ntroduction (Введение) – *Что и почему я изучал?*
- **M**ethods (Методы) – *Как я это изучал?*
- **R**esults (Результаты) – *Что я обнаружил?*
- **a**nd **D**iscussion (Обсуждение) – *Что это значит?*

Часть 2: Композиция научного текста: от замысла к плану (≈1 час)

Задача: Научиться структурировать материал перед написанием.

Практикум "Создание каркаса текста":

- **Тема-пример:** "Сравнение эффективности алгоритмов сортировки данных".
- **Задание:** Коллективно разработать детальный план будущего текста по структуре IMRaD.

1. **Введение:**
   - Актуальность (почему тема важна? Где применяются алгоритмы сортировки?).
   - Проблема (какую конкретную проблему решает работа? Например, выбор оптимального алгоритма для больших данных).
   - Цель работы (сравнить эффективность алгоритмов X, Y, Z).
   - Задачи (1. Проанализировать литературу. 2. Разработать тестовый стенд. 3. Провести эксперимент. 4. Проанализировать результаты.).

2. **Методы (Материалы и методы):**
- Описание алгоритмов (какие алгоритмы выбраны и почему).
- Описание тестового стенда (язык программирования, железо, типы тестовых данных).
- Методика эксперимента (как проводились замеры, какие параметры фиксировались – время, использование памяти).
3. **Результаты:**
- Представление данных в виде таблиц и графиков (без интерпретации!).
- Пример: "График 1 показывает зависимость времени выполнения от объема входных данных для алгоритмов X, Y, Z".
4. **Обсуждение:**
- Интерпретация результатов (почему алгоритм X был быстрее на малых данных, а Y – на больших?).
- Сравнение с данными других исследований (соответствуют ли наши результаты литературным данным?).
- Ограничения исследования (что могло повлиять на чистоту эксперимента?).
5. **Заключение/Выводы:**
- Краткие, пронумерованные ответы на задачи, поставленные во введении.
- Пример: "1. Установлено, что алгоритм X демонстрирует наилучшее быстродействие на массивах объемом до 1000 элементов. 2. ..."

Часть 3: Стилистический практикум: "Перевод с русского на научный" (≈1 час 15 минут)

Задача: Научиться редактировать текст в соответствии с нормами НТС.

Упражнение "Найди и исправь":

Студентам раздаются карточки с текстами, содержащими стилистические ошибки. Задача – найти ошибку и предложить научный вариант.

- **Пример 1 (Разговорность и субъективность):**
  - *Исходник:* «Мы попробовали запустить эту штуку, и она, к нашему удивлению, заработала просто отлично!»
  - *Исправленный вариант:* «Проведенный эксперимент показал высокую эффективность предложенного метода/устройства.»
- **Пример 2 (Неопределенность и неточность):**
  - *Исходник:* «Этот способ дает довольно хорошие результаты при разных условиях.»
  - *Исправленный вариант:* «Предложенный алгоритм позволяет повысить точность распознавания на 15% в условиях зашумленного сигнала.»
- **Пример 3 (Нарушение логики):**
  - *Исходник:* «Было холодно. Датчик показал значение 5°C. Система вышла из строя.»
  - *Исправленный вариант:* «При температуре окружающей среды +5°C была зафиксирована критическая ошибка в работе системы, приведшая к ее отказу.»
- **Пример 4 (Неуместная образность):**
  - *Исходник:* «Данные росли как на дрожжах.»
  - *Исправленный вариант:* «Наблюдался экспоненциальный рост объема данных.»

Практикум по использованию пассивных конструкций:

- *Активная конструкция (нежелательно):* «*Мы* проанализировали полученные спектры.»
- *Пассивная конструкция (предпочтительно):* «Полученные спектры *были проанализированы*.»

Часть 4: Практическое написание и взаимное рецензирование (≈1 час)

Задача: Применить полученные знания для создания короткого текста и оценить работу коллег.

1. **Индивидуальная работа (30 минут):**
   - **Задание:** Используя план, разработанный в Части 2, напишите **Раздел "Методы"** для гипотетической статьи по теме "Сравнение алгоритмов сортировки". Объем: 150-200 слов.
   - Требования: точность, объективность, использование терминов, пассивных конструкций.
2. **Работа в парах / Взаимное рецензирование (30 минут):**
   - Студенты обмениваются написанными текстами.
   - Получают **"Чек-лист рецензента"**:
     - **Композиция:** Есть ли четкое описание объектов, методов, условий эксперимента?
     - **Стиль:** Нет ли разговорных, эмоциональных слов? Преобладают ли пассивные конструкции?
     - **Точность:** Все ли термины использованы корректно? Можно ли воспроизвести эксперимент по этому описанию?
     - **Язык:** Нет ли грамматических и пунктуационных ошибок?
   - Задача рецензента – дать конструктивную письменную обратную связь.

Итоговое задание для самостоятельного выполнения:

Задание: Написать развернутый реферат или введение к научной статье по выбранной на занятии 11-12 теме (объем 2-3 страницы).

Требования к работе:

1. **Структура:** Текст должен содержать логически завершенные разделы (например, Введение, Основная часть, Заключение/Выводы).
2. **Стиль:** Соблюдение всех норм научно-технического стиля (точность, объективность, логичность).
3. **Оформление:** Использование стилей Word для заголовков, основного текста, списка литературы. Наличие автоматического оглавления.
4. **Цитирование:** Корректное использование и оформление цитат и ссылок на источники из ранее составленного библиографического списка.

Критерии оценки:

- **Содержание (40%):** Глубина раскрытия темы, логичность и последовательность изложения.
- **Стиль и язык (40%):** Соответствие нормам НТС, терминологическая точность, грамотность.
- **Оформление (20%):** Соблюдение структурных требований, правильность цитирования, аккуратность.

Это занятие позволит студентам не просто механически оформить текст, а осознанно подойти к процессу научного письма как к инструменту коммуникации в профессиональной среде.

Практическое занятие 17-18: Преобразование документов в HTML/XML

Цель занятия: Освоить принципы и инструменты преобразования документов из стандартных форматов (DOCX, PDF) в структурированные документы, размеченные с помощью языков HTML и XML. Понять разницу между представлением и структурой данных.

Программное обеспечение:

- Текстовый процессор (Microsoft Word, LibreOffice Writer)
- Браузер (Chrome, Firefox)
- Редактор кода (VS Code, Notepad++, Sublime Text) или простой текстовый редактор
- Специализированные конвертеры (например, Pandoc — по желанию, для продвинутых)

Необходимые знания: Базовое понимание структуры документа, работа со стилями в Word, общее представление о том, что такое HTML/XML.

План занятия и Ход работы

Часть 1: Теоретический фундамент. HTML и XML — язык структуры, а не внешнего вида (≈45 минут)

Задача: Понять философию языков разметки.

1. Зачем нужно преобразование?

- **Публикация в веб:** Преобразование отчета, статьи в веб-страницу.
- **Межплатформенный обмен:** XML — универсальный формат для обмена данными между разными системами.
- **Создание электронных архивов:** Структурированные данные (XML) легче индексировать и искать, чем неструктурированные (PDF).
- **Автоматизация процессов:** Данные в XML можно автоматически обрабатывать другими программами.

2. HTML (HyperText Markup Language) vs. XML (eXtensible Markup Language):

- **HTML:** Фиксированный набор тегов (<h1>, <p>, <ul>) для отображения информации в браузере. **Акцент на представлении.**
- **XML:** Позволяет создавать собственные теги (<author>, <price>, <manufacturer>). **Акцент на структуре и смысле данных.**

3. Ключевая концепция: Разделение содержания и представления.

- В Word: *«Это заголовок, потому что он выделен жирным 16м шрифтом»*.
- В HTML/XML: <h1>Это заголовок</h1> — а как его отображать (жирный, 16pt), решает CSS (для HTML) или другое приложение (для XML).

Часть 2: Простое преобразование Word -> HTML (≈1 час 30 минут)

Задача: Освоить базовые методы конвертации и понять их ограничения.

1. Подготовка документа-донора:

- Создайте в Word документ "source_document.docx", используя стили!
  - Заголовок 1: "Мой первый преобразованный документ"
  - Заголовок 2: "Введение"
  - Несколько абзацев основного текста.
  - Нумерованный или маркированный список.
  - Простая таблица (2x2).
  - Рисунок.

2. Метод 1: Сохранение как веб-страницы (Просто, но "грязно").

- **Файл -> Сохранить как -> Веб-страница (*.html)**
- Откройте полученный файл в браузере и в текстовом редакторе.
- **Анализ результата:**
  - В браузере: выглядит похоже на оригинал.
  - В коде: огромное количество служебного кода, стилей style="..." (т.н. "инлайн-стили"). Код нечитаемый и не соответствует современным стандартам веб-разработки.

3. Метод 2: Копирование в онлайн-редактор (Быстро и чище).

- Скопируйте содержимое из Word.
- Войдите в онлайн-редактор, например, **CKEditor** или **TinyMCE** (демо-версии доступны на их сайтах).
- Вставьте скопированный текст в визуальный редактор.
- Переключитесь в режим просмотра исходного кода (HTML).
- **Анализ результата:** Код будет гораздо чище. Он будет использовать семантические теги (<h1>, <p>, <ul>, <table>). Это более правильный подход.

4. Метод 3: Ручное оформление и сохранение (Идеально для контроля).

- Убедитесь, что в документе Word все элементы оформлены СТИЛЯМИ (Заголовок 1, Заголовок 2 и т.д.).
- **Файл -> Сохранить как -> Веб-страница в формате MHTML (***.mht**)** или **"Веб-страница с одним файлом"**.
- Этот метод часто дает более предсказуемый результат, так как четко привязывает структуру Word к HTML-тегам.

Часть 3: Создание и преобразование в пользовательский XML (≈1 час 15 минут)

Задача: Понять, как описывать произвольные данные с помощью XML.

1. Разработка XML-схемы (словаря) для каталога книг.

- Коллективно придумайте структуру для описания книги.
- *Пример:*

```
xml
<catalog>
  <book id="1">
    <title>Война и мир</title>
    <author>Лев Толстой</author>
    <genre>Роман-эпопея</genre>
    <year>1869</year>
    <price currency="RUB">750</price>
  </book>
</catalog>
```

- Обсуждение: Что такое атрибут (id, currency) vs. элемент (<title>).

2. Практикум "Данные -> XML":

- **Задание:** Создайте в Word таблицу с 3-4 книгами (столбцы: Название, Автор, Год, Цена).
- **Метод 1 (Ручной):** "Преобразуйте" эту таблицу, вручную набрав по ней XML-документ в текстовом редакторе. Сохраните как catalog_manual.xml.
- **Метод 2 (Автоматический, используя Word):**

1. Сохраните документ Word как **"Word XML-документ (***.xml**)"**.
2. Откройте полученный файл в текстовом редакторе.
3. **Анализ:** Вы увидите сложнейший XML с огромным количеством служебных тегов Word (<w:document>, <w:body>, <w:p>). Это пример *"XML заданного приложения"* (в данном случае — Microsoft Word). Он нечитаем для человека, но идеален для программной обработки и восстановления документа в Word.

3. Сравнительный анализ:

- Сравните catalog_manual.xml и XML, сгенерированный Word.

- Заполните таблицу:

Критерий	Пользовательский XML (catalog_manual)	XML Word
Читаемость для человека	Высокая	Низкая
Размер файла	Маленький	Большой
Семантическая ценность	Высокая (теги отражают суть данных)	Низкая (теги отражают формат)
Универсальность	Высокая	Только для Word

Часть 4: Валидация и простое преобразование XSLT (≈45 минут)

Задача: Понять, как убедиться в корректности XML и как его можно преобразовать.

1. Валидация XML (Проверка "грамотности").

- Откройте рукописный файл catalog_manual.xml в браузере.
- Браузер — простейший валидатор. Он покажет ошибку, если, например, вы забыли закрыть тег.

2. Введение в XSLT (eXtensible Stylesheet Language Transformations).

- **Идея:** XSLT — это язык-инструкция, который говорит процессору: "возьми этот XML и преобразуй его в другой формат (например, HTML)".

3. Демонстрация "Волшебства XSLT":

- Преподаватель показывает заранее подготовленные файлы:
  - catalog.xml (как выше).
  - transform_to_html.xslt (файл с инструкциями).
- Запускает преобразование (можно использовать онлайн-инструменты или браузер).
- **Результат:** На выходе получается готовый HTML-файл, который отображает каталог книг в виде красивой таблицы на веб-странице.
- **Вывод:** Одни и те же данные (XML) можно по-разному представлять (HTML, PDF, другой XML), используя разные XSLT-преобразования.

Итоговое задание для самостоятельного выполнения:

Сценарий: Вы получили задание подготовить данные для публикации на корпоративном сайте и для передачи в архив в структурированном виде.

Задание:

1. **Подготовка данных:** Создайте в Word документ, содержащий каталог продукции (не менее 5 позиций). Используйте таблицу. Колонки: "Наименование", "Артикул", "Характеристики", "Стоимость". Обязательно используйте Стили для заголовка документа.
2. **Преобразование в HTML:** Преобразуйте этот документ в HTML двумя способами. Сравните полученные HTML-файлы по критериям: чистота кода, размер, соответствие исходной структуре.
3. **Преобразование в XML:**
   - Разработайте свою собственную XML-схему для описания товара.
   - Вручную создайте XML-файл products.xml, описав в нем товары из вашего каталога в соответствии с вашей схемой.
4. **Отчет:** Напишите краткий отчет (1-2 стр.), в котором:

- Опишите преимущества и недостатки каждого использованного метода преобразования.
- Объясните, почему для обмена данными между предприятиями предпочтительнее использовать XML, а не DOCX или PDF.

Критерии успешного выполнения:

- Корректность и чистота сгенерированного HTML-кода.
- Логичность и валидность созданной XML-структуры.
- Глубина анализа в отчете, понимание разницы между форматами.

Практическое занятие 19-20: Составление несложных стилей на языке Cascade Style Sheets (CSS) и отлаживание их

Цель занятия: Научиться создавать и применять каскадные таблицы стилей (CSS) для управления внешним видом HTML-документов. Освоить базовые принципы отладки стилей в браузере.

Программное обеспечение:

- Редактор кода (VS Code, Sublime Text, Notepad++)
- Современный браузер (Chrome, Firefox, Edge) с инструментами разработчика
- Текстовый процессор для ведения записей

Необходимые знания: Базовое понимание структуры HTML (теги, атрибуты, вложенность).

План занятия и Ход работы

Часть 1: Введение в CSS. Отделение содержания от оформления (≈45 минут)

Задача: Понять философию CSS и базовый синтаксис.

1. Зачем нужен CSS?

- **До CSS:** Оформление было внутри HTML (<font color="red">), что делало код громоздким и негибким.
- **После CSS:** HTML отвечает за **структуру** и **содержание**, CSS — за **внешний вид**.

2. Три способа подключения CSS к HTML:

- **Внешняя таблица стилей (рекомендуется):**

```
<link rel="stylesheet" href="styles.css">
```

- **Внутренние стили (в <head>):**

```
<style>
  p { color: blue; }
</style>
```

- **Встроенные стили (атрибут style):**

```
<p style="color: red;">Текст</p>
```

3. Базовый синтаксис CSS:

```
селектор {
  свойство: значение;
  свойство: значение;
}
```

- **Селектор** - указывает, к какому элементу HTML применить стиль
- **Свойство** - что именно мы хотим изменить (цвет, размер, отступ)
- **Значение** - как именно мы это изменяем

Часть 2: Практикум по созданию HTML-документа и подключению CSS (≈30 минут)

Задача: Создать основу для дальнейшей работы.

1. Создание файловой структуры:

- Создайте папку css-practice
- Внутри создайте файлы:
  - index.html - основная HTML-страница
  - style.css - таблица стилей

2. Наполнение HTML-документа:

```
<!DOCTYPE html>
<html lang="ru">
<head>
  <meta charset="UTF-8">
  <title>Мой первый стилизованный документ</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <header>
    <h1>Технологии будущего</h1>
  </header>
  <nav>
    <ul>
      <li><a href="#">Искусственный интеллект</a></li>
      <li><a href="#">Квантовые вычисления</a></li>
      <li><a href="#">Биотехнологии</a></li>
    </ul>
  </nav>
```

```
        <main>
        <article>
            <h2>Нейронные сети в медицине</h2>
            <p>Современные нейронные сети позволяют диагностировать заболевания с точностью, превышающей человеческие возможности.</p>
            <p class="important">Особенно эффективны они в анализе медицинских изображений.</p>
        </article>
            <aside>
            <h3>Интересный факт</h3>
            <p>Первая нейронная сеть была создана в 1957 году.</p>
        </aside>
    </main>
        <footer>
        <p>&copy; 2023 Научный портал</p>
    </footer>
</body>
</html>
```

Часть 3: Основные селекторы и свойства CSS (≈1 час 30 минут)

Задача: Освоить основные механизмы выбора элементов и управления их внешним видом.

Практическое задание "Постепенное оформление":

1. Селекторы по тегам:

```
/* Стили для тела документа */
body {
    font-family: Arial, sans-serif;
    line-height: 1.6;
    margin: 0;
    padding: 0;
}
/* Стили для всех заголовков первого уровня */
h1 {
    color: #2c3e50;
    text-align: center;
}
/* Стили для всех параграфов */
p {
    margin-bottom: 15px;
}
```

2. Селекторы по классам:

```
* Стиль для важных элементов */
.important {
    background-color: #fffacd;
    padding: 10px;
    border-left: 3px solid #ffd700;
}

/* Стиль для навигационных ссылок */
.nav-link {
    text-decoration: none;
    color: #3498db;
}
```

3. Селекторы по идентификаторам:

```
/* Стиль для главного заголовка */
#main-title {
    font-size: 2.5em;
    text-shadow: 2px 2px 4px rgba(0,0,0,0.3);
}
```

4. Блочная модель CSS (Box Model):

```
/* Практика работы с отступами и границами */
article {
    background-color: #f9f9f9;
    padding: 20px;
    margin: 20px;
    border: 1px solid #ddd;
    border-radius: 5px;
}
nav ul {
    background-color: #34495e;
    padding: 15px;
    margin: 0;
}
nav li {
    display: inline-block;
    margin-right: 20px;
}
```

5. Псевдоклассы:

```
/* Изменение стиля ссылок при наведении */
a:hover {
    color: #e74c3c;
    text-decoration: underline;
}

/* Стиль для первой буквы параграфа */
p::first-letter {
    font-size: 1.5em;
    font-weight: bold;
    color: #2980b9;
}
```

Часть 4: Отладка CSS с помощью инструментов разработчика (≈45 минут)

Задача: Научиться находить и исправлять ошибки в стилях.

Практикум "Детектив CSS":

1. Открытие инструментов разработчика:

- Нажмите F12 или правой кнопкой → "Исследовать элемент"

2. Упражнение "Найди проблему":

- Преподаватель дает HTML-страницу с "сломанными" стилями
- Задачи для студентов:
  - Почему заголовок не центрируется?
  - Почему ссылки не меняют цвет при наведении?
  - Почему отступы выглядят не так, как ожидалось?

3. Работа с вкладками инструментов разработчика:

- **Elements/Inspector:** Просмотр HTML-структуры и примененных стилей
- **Styles:** Просмотр и редактирование CSS в реальном времени
- **Computed:** Анализ итоговых вычисленных стилей
- **Box Model:** Визуализация отступов, границ и полей

4. Практическое задание по отладке:

```
/* CSS с преднамеренными ошибками */
.header {
    background-color: blue;
    text-allign: center; /* Ошибка в написании */
}
.navigation a {
    color: black;
    text-decoration: none
    /* Отсутствует точка с запятой */
}
.content {
    width: 800px
    margin: 0 auto;
    /* Неправильная работа с шириной и отступами */
}
```

Методика отладки:

1. Открыть инструменты разработчика
2. Найти проблемный элемент
3. Проверить, какие стили применяются
4. Найти перечеркнутые свойства (не применяются)
5. Проверить синтаксис
6. Исправить ошибку в исходном CSS-файле

Часть 5: Создание адаптивного макета (≈30 минут)

Задача: Познакомиться с основами адаптивной верстки.

```
/* Базовые стили для мобильных устройств */
.container {
    width: 100%;
    padding: 10px;
}
/* Медиа-запрос для планшетов */
@media (min-width: 768px) {
    .container {
        width: 750px;
        margin: 0 auto;
    }
    main {
        display: flex;
    }

    article {
        flex: 2;
    }
    aside {
        flex: 1;
    }
```

```
}
/* Медиа-запрос для десктопов */
@media (min-width: 1200px) {
  .container {
    width: 1170px;
  }
}
```

Итоговое задание для самостоятельного выполнения:

Задание: Создать стилизованную веб-страницу портфолио.

Требования:

1. **HTML-структура:** Использовать семантические теги (header, nav, main, section, footer)
2. **CSS-оформление:**
   - Использовать все три типа селекторов (по тегу, классу, ID)
   - Применить минимум 10 различных CSS-свойств
   - Реализовать hover-эффекты для интерактивных элементов
   - Создать адаптивный макет с помощью медиа-запросов
3. **Содержание:**
   - Страница должна содержать информацию о себе (реальной или вымышленной)
   - Разделы: "Обо мне", "Навыки", "Проекты", "Контакты"

Критерии оценки:

- **Качество кода (40%):** Чистота и правильность CSS-синтаксиса, логичная организация стилей
- **Визуальное оформление (30%):** Единообразие стиля, гармоничная цветовая схема
- **Функциональность (20%):** Корректная работа всех стилей, включая hover-эффекты
- **Адаптивность (10%):** Корректное отображение на разных размерах экрана

Пример структуры проекта:

```
text
my-portfolio/
├── index.html
├── styles/
│   └── main.css
└── images/
    └── avatar.jpg
```

Это занятие даст студентам практические навыки работы с CSS - основным инструментом визуального оформления в веб-разработке.

Практическое занятие 21-22: Преобразование рисунков в форматы PNG и JPEG

Цель занятия: Освоить принципы преобразования графических изображений, включая векторные форматы, в растровые форматы PNG и JPEG. Научиться выбирать оптимальный формат и настройки сжатия в зависимости от задач.

Программное обеспечение:

- Растровые редакторы: GIMP (бесплатный), Adobe Photoshop (платный)
- Просмотрщики изображений: IrfanView, XnView
- Векторные редакторы: Inkscape (бесплатный), Adobe Illustrator (платный)
- Онлайн-конвертеры (как дополнительный инструмент)

Необходимые знания: Базовое понимание различий между растровой и векторной графикой.

План занятия и Ход работы

Часть 1: Теоретические основы растровой и векторной графики (≈45 минут)

Задача: Понять фундаментальные различия между типами графики.

1. Векторная графика:

- **Принцип:** Изображение описывается математическими формулами (геометрическими примитивами - линии, кривые, фигуры).
- **Форматы:** SVG, AI, EPS, CDR
- **Преимущества:**
  - Масштабируемость без потери качества
  - Маленький размер файла для простых изображений
  - Легкое редактирование элементов
- **Недостатки:**
  - Не подходит для фотореалистичных изображений
  - Сложность создания сложных детализированных изображений

2. Растровая графика:

- **Принцип:** Изображение состоит из сетки пикселей (точек), каждая из которых имеет свой цвет.
- **Форматы:** JPEG, PNG, GIF, BMP, TIFF
- **Преимущества:**
  - Фотореалистичное изображение
  - Богатая цветовая палитра
  - Подходит для сложных текстур и градиентов
- **Недостатки:**
  - Потеря качества при увеличении
  - Большой размер файла для высококачественных изображений

3. Ключевые понятия:

- **Разрешение:** Количество пикселей на единицу длины (DPI - dots per inch)
- **Глубина цвета:** Количество бит для кодирования цвета одного пикселя
- **Сжатие:** Алгоритмы уменьшения размера файла (с потерями/без потерь)

Часть 2: Форматы PNG и JPEG - сравнительный анализ (≈30 минут)

Задача: Научиться выбирать правильный формат для конкретной задачи.

Параметр	JPEG (JPG)	PNG
Тип сжатия	С потерями	Без потерь
Поддержка прозрачности	Нет	Да (альфа-канал)
Количество цветов	16,7 млн (24 бит)	До 16,7 млн (24 бит) + 256 уровней прозрачности
Лучшее применение	Фотографии, изображения с плавными переходами цветов	Логотипы, скриншоты, изображения с текстом, графики с резкими границами цветов
Размер файла	Меньше (при сравнимом визуальном качестве)	Больше

Практическое правило выбора:

- **Используйте JPEG:** для фотографий, художественных изображений
- **Используйте PNG:** когда нужна прозрачность, резкие границы, текст

Часть 3: Практикум преобразования векторных форматов в PNG/JPEG (≈1 час 30 минут)

Задача: Освоить процесс конвертации векторных изображений.

1. Подготовка исходных материалов:

- Создайте или скачайте несколько векторных изображений:
  - Простой логотип (SVG)
  - Техническая схема (SVG)
  - Иллюстрация с градиентами (AI или EPS)

2. Метод 1: Конвертация в Inkscape

- Откройте векторный файл в Inkscape
- **Файл → Сохранить как**
- Для **PNG:**
  - Выберите "Portable Network Graphics (*.png)"
  - Настройки экспорта:
    - Разрешение: 300 DPI (для печати) или 96 DPI (для веб)
    - Ширина/Высота: задайте нужные размеры в пикселях
    - Антиалиазинг: включить для сглаживания краев
- Для **JPEG:**
  - Inkscape не поддерживает прямой экспорт в JPEG
  - Сначала экспортируйте в PNG, затем конвертируйте в GIMP

3. Метод 2: Пакетная обработка в XnView

- Откройте XnView
- **Инструменты → Пакетная обработка**
- Добавьте несколько векторных файлов
- Настройте преобразование:
  - Выходной формат: PNG или JPEG
  - Качество для JPEG: 80%
  - Размер: задайте нужные параметры

4. Упражнение "Сравнение результатов":

- Конвертируйте один векторный файл в PNG и JPEG с разными настройками
- Сравните:
  - Размер файла
  - Качество изображения при увеличении
  - Наличие артефактов

Часть 4: Работа с растровыми изображениями в GIMP (≈1 час 30 минут)

Задача: Освоить тонкую настройку параметров экспорта.

1. Базовые операции:

- Изменение размера изображения (**Изображение → Размер изображения**)
- Обрезка (**Инструмент обрезки**)
- Коррекция цветов (**Цвета → Яркость-Контраст**)

2. Экспорт в PNG:

- Откройте изображение в GIMP
- **Файл → Экспортировать как**
- Выберите формат PNG
- Дополнительные параметры:
  - **Сжатие уровня 9 -** максимальное сжатие без потерь
  - **Сохранить цветовой профиль -** для профессиональной работы с цветом
  - **Сохранить слои -** только если нужно сохранить структуру

3. Экспорт в JPEG:

- **Файл → Экспортировать как** → выберите JPEG
- Появится диалоговое окно настроек:
  - **Качество:** от 0 (худшее) до 100 (лучшее)
    - Для веб: 70-85%
    - Для печати: 90-100%
  - **Сглаживание:** улучшает качество при высоком сжатии

- **Оптимизировать:** уменьшает размер файла
- **Прогрессивный:** изображение загружается постепенно

4. Практическое задание "Оптимизатор":

- Возьмите фотографию и экспортируйте ее в JPEG с разными настройками качества:
  - 100% - эталонное качество
  - 85% - хороший баланс
  - 50% - заметная потеря качества
  - 25% - сильные артефакты
- Создайте таблицу сравнения:

Качество	Размер файла	Визуальная оценка	Применение
100%	2.5 МБ	Отличное	Полиграфия
85%	450 КБ	Хорошее	Веб-сайты
50%	150 КБ	Удовлетворительное	Электронная почта
25%	75 КБ	Плохое	Миниатюры

Часть 5: Специальные случаи и оптимизация (≈45 минут)

Задача: Научиться решать нестандартные задачи оптимизации.

1. Создание изображений с прозрачностью:

- В GIMP: создайте изображение с прозрачным фоном
- Добавьте текст или фигуры
- Экспортируйте в PNG
- Проверьте, что фон остался прозрачным

2. Пакетная обработка скриншотов:

- Сделайте несколько скриншотов (они обычно в PNG)
- В IrfanView: **Файл → Пакетная обработка/переименование**
- Настройте конвертацию в JPEG с оптимальным качеством

3. Оптимизация для веб:

- Используйте инструменты типа **ImageOptim** или онлайн-сервисы
- Они дополнительно сжимают PNG без потери качества
- Сравните размеры до и после оптимизации

4. Создание favicon:

- Создайте изображение 16x16 или 32x32 пикселя
- Экспортируйте в PNG
- Конвертируйте в ICO с помощью онлайн-конвертера

Итоговое задание для самостоятельного выполнения:

Сценарий: Вы - веб-разработчик, которому нужно подготовить графику для сайта компании.

Задание:

1. **Подготовить логотип:**
   - Взять векторный логотип (SVG)
   - Конвертировать в PNG с прозрачным фоном (3 размера: 32x32, 128x128, 512x512)
   - Оптимизировать размер файлов
2. **Обработать галерею фотографий:**
   - Взять 5 фотографий в высоком разрешении
   - Создать три версии для каждой:
     - Полноразмерная (для детального просмотра, JPEG 90%)
     - Средняя (для галереи, JPEG 80%)
     - Миниатюра (для превью, JPEG 70%)
3. **Создать баннер:**
   - Создать баннер 800x200 пикселей
   - Использовать прозрачность и градиенты
   - Сохранить в PNG-24
4. **Отчет:**
   - Предоставить все полученные файлы
   - Составить таблицу с исходными и итоговыми размерами файлов
   - Обосновать выбор форматов и настроек для каждого типа изображений

Критерии оценки:

- **Техническая корректность (40%):** Правильность выбора форматов, корректность настроек экспорта
- **Качество изображений (30%):** Отсутствие артефактов, сохранение важных деталей
- **Оптимизация (20%):** Эффективное соотношение качество/размер файла
- **Организация работы (10%):** Логичная структура файлов, четкий отчет

Это занятие даст студентам практические навыки, которые immediately применимы в веб-разработке, дизайне и подготовке материалов для полиграфии.

Практическое занятие 23-24: Обнаружение и устранение ошибок разметки

Цель занятия: Научиться выявлять, анализировать и исправлять синтаксические и структурные ошибки в документах с разметкой (HTML, XML). Освоить инструменты валидации и разработать системный подход к отладке.

Программное обеспечение:

- Редактор кода (VS Code с плагинами)
- Валидаторы: W3C Validator, XML Validator
- Браузеры с инструментами разработчика
- Специализированные утилиты (HTML Tidy, xmllint)

Необходимые знания: Базовое понимание HTML/XML синтаксиса, работа с CSS.

План занятия и Ход работы

Часть 1: Теоретические основы валидации (≈45 минут)

Задача: Понять важность корректной разметки и типы ошибок.

1. Зачем нужна валидация?

- **Кроссбраузерность:** Корректное отображение во всех браузерах
- **Доступность:** Правильная работа скринридеров и вспомогательных технологий
- **SEO:** Лучшая индексация поисковыми системами
- **Совместимость:** Корректная работа с CSS и JavaScript
- **Техническое долголетие:** Гарантия работы в будущих версиях браузеров

2. Типы ошибок разметки:

- **Синтаксические:** Нарушение базовых правил языка
- **Структурные:** Неправильная вложенность элементов
- **Семантические:** Неправильное использование тегов по назначению
- **Атрибутные:** Неправильные имена или значения атрибутов

3. Уровни строгости проверки:

- **Well-formed:** Документ соответствует базовому синтаксису
- **Valid:** Документ соответствует спецификации (DTD, Schema)

Часть 2: Инструменты валидации (≈1 час)

Задача: Освоить основные инструменты проверки разметки.

1. Онлайн-валидаторы W3C:

- **HTML Validator:** https://validator.w3.org/
- **CSS Validator:** https://jigsaw.w3.org/css-validator/

2. Встроенные валидаторы редакторов:

- **VS Code:** Плагины "HTMLHint", "XML Tools"
- **Демонстрация:** Показ работы плагинов в реальном времени

3. Браузерные инструменты:

- **Консоль ошибок:** Показывает критические ошибки
- **Инспектор элементов:** Визуализация структуры DOM

4. Практическое задание "Первое знакомство":

- Дать студентам заведомо невалидный HTML
- Попросить проверить через W3C Validator
- Проанализировать вывод валидатора

```
<!DOCTYPE html>
<html>
<head>
  <title>Тестовая страница с ошибками</title>
</head>
<body>
  <h1>Заголовок</h2>
  <p>Текст без закрывающего тега
  <div><span>Неправильная вложенность</div></span>
  <img src="image.jpg">
  <a href="page.html">Ссылка</a>
</body>
</html>
```

Часть 3: Типичные ошибки HTML и методы их исправления (≈2 часа)

Задача: Научиться диагностировать и исправлять распространенные ошибки.

Практикум "Классификация ошибок":

1. Синтаксические ошибки:

```
<!-- Незакрытые теги -->
<p>Этот параграф не закрыт
<!-- Неправильное закрытие -->
<strong><em>Текст</strong></em>

<!-- Непарные теги -->
```

```
<br></br>  <!-- Лишний закрывающий тег -->
```
Метод исправления: Проверка парности всех тегов

2. Структурные ошибки:

```
<!-- Неправильная вложенность блочных и строчных элементов -->
<p>Текст <div>блочный элемент внутри строчного</div> продолжение</p>

<!-- Нарушение иерархии -->
<ul>
  <li>Пункт 1</li>
  <p>Абзац внутри списка</p>
  <li>Пункт 2</li>
</ul>
```
Метод исправления: Анализ DOM-дерева в инструментах разработчика

3. Семантические ошибки:

```
<!-- Использование тегов не по назначению -->
<div onclick="goToPage()">Нажми меня</div>  <!-- Должна быть кнопка или ссылка -->
<!-- Неправильные заголовки -->
<h3>Главный заголовок</h3>  <!-- Должен быть h1 -->
<h1>Подраздел</h1>          <!-- Нарушена иерархия -->
```
Метод исправления: Изучение спецификации HTML5

4. Атрибутные ошибки:

```
<!-- Несуществующие атрибуты -->
<div class="header" id="main" style="color:red" randomattr="value">
<!-- Неправильные значения -->
<input type="text" required="true">  <!-- Должно быть required или required="" -->
<!-- Отсутствующие обязательные атрибуты -->
<img>  <!-- Нет src и alt -->
```
Метод исправления: Проверка допустимых атрибутов для каждого элемента

Часть 4: Работа с XML-документами (≈1 час 30 минут)

Задача: Освоить особенности валидации XML.

1. Проверка well-formedness:

- Правильное закрытие всех тегов
- Корректная вложенность
- Экранирование специальных символов

2. Валидация против DTD/XSD:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE catalog SYSTEM "catalog.dtd">
<catalog>
  <book id="1">
    <title>Война и мир</title>
    <author>Лев Толстой</author>
  </book>
</catalog>
```
3. Практическое задание "XML-детектив":

- Студентам дается XML с ошибками
- Задача: найти и классифицировать ошибки

```
<?xml version="1.0" encoding="UTF-8"?>
<library>
  <book id=1>  <!-- Ошибка: значение атрибута без кавычек -->
    <title>Преступление и наказание<title>  <!-- Ошибка: опечатка в закрывающем теге -->
    <author>Фёдор Достоевский</author>
  <book>       <!-- Ошибка: незакрытый тег -->
  <movie>      <!-- Ошибка: элемент не описан в DTD -->
    <title>Матрица</title>
  </movie>
</library>
```
Часть 5: Системный подход к отладке (≈1 час)

Задача: Разработать методику поиска и исправления ошибок.

Алгоритм отладки:

1. Подготовка:

- Убедиться, что документ имеет правильную кодировку
- Проверить объявление DOCTYPE

2. Поэтапная проверка:

```
<!DOCTYPE html>
<html lang="ru">
<head>
  <meta charset="UTF-8">
  <title>Поэтапная отладка</title>
</head>
<body>
  <!-- Шаг 1: Проверить базовую структуру -->
  <header>Шапка</header>
  <!-- Шаг 2: Добавить контент и проверить -->
  <main>
    <article>
      <h1>Заголовок</h1>
      <p>Абзац текста</p>
```

```
    </article>
  </main>

  <!-- Шаг 3: Добавить сложные элементы -->
  <footer>
    <nav>
      <ul>
        <li><a href="#link1">Ссылка 1</a></li>
      </ul>
    </nav>
  </footer>
</body>
</html>
```

3. Инструментарий отладчика:

- **Валидатор W3C:** Для комплексной проверки
- **HTML Tidy:** Для автоматического исправления
- **XML Lint:** Для проверки XML
- **Browser Console:** Для выявления runtime-ошибок

4. Практическое задание "Рефакторинг":

- Студентам дается "грязный" HTML-код
- Задача: привести к валидному состоянию

```
<HTML>
<HEAD>
<TITLE>Плохой HTML</TITLE>
</HEAD>
<BODY BGCOLOR="white">
<CENTER><H1>Центрированный заголовок</H1></CENTER>
<P><FONT COLOR="red">Красный текст</FONT>
<TABLE WIDTH="100%">
<TR><TD>Ячейка 1<TD>Ячейка 2
</TABLE>
<SCRIPT LANGUAGE="JavaScript">
document.write("Привет мир")
</SCRIPT>
</BODY>
</HTML>
```

Итоговое задание для самостоятельного выполнения:

Сценарий: Вы получили от коллеги HTML-документ, который некорректно отображается в браузерах.

Задание:

1. **Анализ:** Провести полный аудит документа на наличие ошибок
2. **Классификация:** Составить таблицу найденных ошибок с указанием:
   - Типа ошибки
   - Местоположения
   - Серьезности
   - Предложенного исправления
3. **Исправление:** Создать исправленную версию документа
4. **Верификация:** Убедиться, что исправленный документ проходит валидацию

Исходный документ для анализа:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>Документ с ошибками</title>
  <style>
    .header { color: blue; }
  </style>
</head>
<body>
  <div class="header">
    <h1>Главная страница</h1>
  </div>
  <main>
    <section id="content">
      <h2>О компании</h2>
      <p>Мы лучшие в своем деле
      <p>Наша команда профессионалов</p>
      <ul>
        <li>Качество
        <li>Скорость
        <li>Надежность
      </ul>
    </section>
    <aside>
      <h3>Новости<h3>
      <article>
        <h4>Важная новость</h4>
```

```
            <p>Текст новости...</p>
          <article>
        </aside>
    </main>
    <footer>
        <p>Контакты: <a href="mailto:info@company.com">info@company.com</a></p>
        <p>&copy; 2023 Все права защищены
    </footer>
    <script>
        function showMessage() {
            alert('Добро пожаловать!');
        }
        showMessage();
    </script>
</body>
</html>
```

Критерии оценки:

- **Полнота анализа (40%):** Выявлены все типы ошибок
- **Качество исправлений (30%):** Корректность предложенных решений
- **Методология (20%):** Системный подход к отладке
- **Валидность результата (10%):** Исправленный документ проходит проверку

Это занятие сформирует у студентов критически важный навык - умение находить и исправлять ошибки в разметке, что является основой профессиональной веб-разработки.

Практическое занятие 25-26: Придание разметке документа удобочитаемого структурированного вида

Цель занятия: Научиться преобразовывать "сырую" разметку в хорошо структурированный, легко читаемый и поддерживаемый код. Освоить принципы форматирования, организации и документирования HTML/XML документов.

Программное обеспечение:

- Редактор кода (VS Code, Sublime Text)
- Инструменты автоматического форматирования (Prettier, HTML Tidy)
- Браузер с инструментами разработчика
- Валидаторы разметки

Необходимые знания: Базовое понимание HTML/XML синтаксиса, опыт обнаружения ошибок разметки.

План занятия и Ход работы

Часть 1: Философия читаемого кода (≈45 минут)

Задача: Понять важность форматирования кода для командной работы и поддержки.

1. Зачем нужно форматирование?

- **Читаемость:** Легкое понимание структуры документа
- **Поддерживаемость:** Упрощение внесения изменений
- **Отладка:** Быстрое нахождение ошибок
- **Командная работа:** Единый стандарт для всех разработчиков
- **Профессионализм:** Признак качественной работы

2. Принципы хорошего форматирования:

- **Последовательность:** Единый стиль во всем документе
- **Иерархичность:** Визуальное отражение структуры документа
- **Семантичность:** Ясное понимание назначения каждого элемента
- **Минимализм:** Отсутствие избыточных пробелов и комментариев

Часть 2: Базовые правила форматирования HTML (≈1 час 30 минут)

Задача: Освоить практические приемы форматирования HTML-кода.

1. Отступы и выравнивание:

```
<!-- ПЛОХО: нет отступов -->
<div><h1>Заголовок</h1><p>Текст</p><ul><li>Пункт 1</li><li>Пункт 2</li></ul></div>

<!-- ХОРОШО: четкая иерархия -->
<div>
    <h1>Заголовок</h1>
    <p>Текст</p>
    <ul>
        <li>Пункт 1</li>
        <li>Пункт 2</li>
    </ul>
</div>
```

2. Размещение атрибутов:

```
<!-- ПЛОХО: все в одну строку -->
<img src="image.jpg" alt="Описание" class="image" id="main-image" data-info="value">
<!-- ХОРОШО: каждый атрибут на новой строке при большом количестве -->
<img
    src="image.jpg"
```

```
  alt="Описание"
  class="image"
  id="main-image"
  data-info="value"
>
```

3. Практическое задание "Реформатор":

- Студентам дается неформатированный HTML-код
- Задача: применить правила форматирования

Исходный код:

```
<!DOCTYPE html><html lang="ru"><head><meta charset="UTF-8"><title>Неформатированный документ</title><style>body{font-family:Arial;margin:0;}</style></head><body><header><h1>Заголовок сайта</h1><nav><ul><li><a href="#home">Главная</a></li><li><a href="#about">О нас</a></li><li><a href="#contact">Контакты</a></li></ul></nav></header><main><section id="content"><article><h2>Статья 1</h2><p>Текст статьи</p></article><article><h2>Статья 2</h2><p>Текст статьи</p></article></section></main><footer><p>Copyright 2023</p></footer></body></html>
```

Часть 3: Продвинутые техники организации кода (≈1 час 30 минут)

Задача: Научиться группировать и организовывать сложные документы.

1. Логическое разделение на секции:

```
<!-- ===== HEADER SECTION ===== -->
<header class="main-header">
  <!-- Навигация -->
  <nav>...</nav>
</header>

<!-- ===== MAIN CONTENT ===== -->
<main class="content">
  <!-- Боковая панель -->
  <aside>...</aside>
  <!-- Основной контент -->
  <article>...</article>
</main>
<!-- ===== FOOTER SECTION ===== -->
<footer class="main-footer">
  <!-- Контактная информация -->
  <address>...</address>
</footer>
```

2. Организация CSS-классов:

```
<!-- ПЛОХО: непонятные названия классов -->
<div class="b1 bx f-left w50"></div>

<!-- ХОРОШО: осмысленные BEM-имена -->
<div class="product-card product-card--featured">
  <img class="product-card__image" src="...">
  <h3 class="product-card__title">Название товара</h3>
  <p class="product-card__description">Описание товара</p>
</div>
```

3. Группировка связанных элементов:

```
<!-- Форма с логической группировкой -->
<form class="registration-form">
  <!-- Персональные данные -->
  <fieldset class="personal-data">
    <legend>Персональная информация</legend>
    <div class="form-group">...</div>
    <div class="form-group">...</div>
  </fieldset>

  <!-- Контактные данные -->
  <fieldset class="contact-data">
    <legend>Контактная информация</legend>
    <div class="form-group">...</div>
  </fieldset>
</form>
```

Часть 4: Комментирование и документация (≈45 минут)

Задача: Научиться добавлять полезные комментарии.

1. Типы комментариев:

```
<!-- ============================================
  HEADER SECTION
  ============================================ -->
<header>
  <!-- Главная навигация -->
  <nav>
    <!-- TODO: Добавить выпадающее меню -->
    <ul>...</ul>
  </nav>

  <!--
    Баннер с акциями
```

```
    Отображается только для зарегистрированных пользователей
  -->
  <div class="promo-banner">...</div>
</header>

<!-- FIXME: Убрать после тестирования -->
<div class="debug-panel">...</div>
```

2. Правила хорошего комментирования:

- Комментируйте **зачем**, а не **что**
- Избегайте избыточных комментариев
- Используйте стандартные метки (TODO, FIXME, NOTE)
- Обновляйте комментарии при изменении кода

3. Практическое задание "Документатор":

- Добавить осмысленные комментарии в 给定的 HTML-документ
- Разметить логические секции
- Добавить TODO-заметки для будущих улучшений

Часть 5: Инструменты автоматического форматирования (≈1 час)

Задача: Освоить использование инструментов для автоматического форматирования.

1. Prettier - настройка и использование:

```
json
// .prettierrc
{
  "printWidth": 80,
  "tabWidth": 4,
  "useTabs": false,
  "semi": true,
  "singleQuote": true,
  "trailingComma": "es5",
  "bracketSpacing": true,
  "arrowParens": "avoid"
}
```

2. HTML Tidy:

```
bash
# Пример использования в командной строке
tidy -i -w 80 -utf8 --indent-spaces 4 input.html > output.html
```

3. Встроенные средства редакторов:

- **VS Code:** Shift + Alt + F
- **Sublime Text:** Edit → Line → Reindent
- **Online-форматтеры:** HTML Formatter, XML Formatter

4. Практическое задание "Автоматизация":

- Настроить Prettier для HTML-форматирования
- Сравнить ручное и автоматическое форматирование
- Создать конфигурацию для командного проекта

Часть 6: Структурирование сложных XML-документов (≈45 минут)

Задача: Применить принципы форматирования к XML-документам.

1. Форматирование XML с пространствами имен:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
  Каталог продукции
  Версия: 1.2
  Дата создания: 2023-10-25
-->
<catalog xmlns:prod="http://example.com/products">
  <!-- Электроника -->
  <category name="electronics">
    <product id="p001" status="available">
      <prod:name>Смартфон Model X</prod:name>
      <prod:price currency="USD">599.99</prod:price>
      <specifications>
        <screen>6.1"</screen>
        <memory>128GB</memory>
      </specifications>
    </product>
  </category>
  <!-- Книги -->
  <category name="books">
    <product id="p002" status="out-of-stock">
      <prod:name>Искусство программирования</prod:name>
      <prod:price currency="RUB">2450.00</prod:price>
    </product>
  </category>
</catalog>
```

Итоговое задание для самостоятельного выполнения:

Сценарий: Вы получили от коллеги "сырой" HTML-код веб-приложения, который нужно привести в читаемый вид.

Задание:

1. **Анализ:** Изучить исходный код и выявить проблемы форматирования
2. **Рефакторинг:** Применить все изученные техники:
   - Расставить отступы и переносы строк
   - Логически сгруппировать элементы
   - Добавить осмысленные комментарии
   - Оптимизировать структуру документа
3. **Документирование:** Создать файл README с описанием примененных правил
4. **Автоматизация:** Настроить инструмент автоматического форматирования

Исходный документ:

```
<!DOCTYPE html><html lang="ru"><head><meta charset="UTF-8"><meta name="viewport" content="width=device-width, initial-scale=1.0"><title>Интернет-магазин</title><style>.header{background:#333;color:white;padding:1rem}.nav-list{list-style:none;margin:0;padding:0;display:flex}.nav-item{margin-right:1rem}.product-grid{display:grid;grid-template-columns:repeat(3,1fr);gap:1rem}.product-card{border:1px solid #ddd;padding:1rem}.btn{background:blue;color:white;padding:0.5rem 1rem;border:none;cursor:pointer}</style></head><body><header class="header"><h1>Магазин техники</h1><nav><ul class="nav-list"><li class="nav-item"><a href="#phones">Смартфоны</a></li><li class="nav-item"><a href="#laptops">Ноутбуки</a></li><li class="nav-item"><a href="#tablets">Планшеты</a></li></ul></nav></header><main><section id="phones"><h2>Смартфоны</h2><div class="product-grid"><div class="product-card"><h3>iPhone 14</h3><p>Последняя модель Apple</p><p class="price">$999</p><button class="btn">Купить</button></div><div class="product-card"><h3>Samsung Galaxy S23</h3><p>Флагман от Samsung</p><p class="price">$899</p><button class="btn">Купить</button></div></div></section><section id="laptops"><h2>Ноутбуки</h2><div class="product-grid"><div class="product-card"><h3>MacBook Pro</h3><p>Для профессионалов</p><p class="price">$1999</p><button class="btn">Купить</button></div></div></section></main><footer><p>Контакты: info@store.com</p><p>© 2023 Магазин техники</p></footer><script>document.querySelectorAll('.btn').forEach(btn=>{btn.addEventListener('click', ()=>{alert('Товар добавлен в корзину')})})</script></body></html>
```

Критерии оценки:

- **Читаемость (40%):** Логичная структура, правильные отступы
- **Организация (30%):** Группировка связанных элементов, семантичные имена
- **Документирование (20%):** Качество и уместность комментариев
- **Соблюдение стандартов (10%):** Единообразие форматирования

Это занятие научит студентов профессиональному подходу к организации кода, что значительно повысит эффективность их работы в реальных проектах.

Практическое занятие 27-28: Создание информативных графических схем

Цель занятия: Научиться создавать понятные, информативные и визуально привлекательные графические схемы для представления сложной информации. Освоить принципы визуальной коммуникации и инструменты создания диаграмм.

Программное обеспечение:

- Векторные редакторы: Draw.io (диаграммы), yEd Graph Editor
- Онлайн-инструменты: Lucidchart, Miro
- Офисные пакеты: Microsoft Visio, LibreOffice Draw
- Специализированные инструменты: PlantUML (для UML-диаграмм)

Необходимые знания: Базовое понимание структуры данных, опыт работы с компьютерной графикой.

План занятия и Ход работы

Часть 1: Принципы эффективной визуализации (≈45 минут)

Задача: Понять основы визуального восприятия и коммуникации.

1. Зачем нужны схемы?

- **Упрощение сложного:** Визуальное представление сложных систем
- **Быстрое понимание:** Схема воспринимается быстрее текста
- **Коммуникация:** Единое понимание в команде
- **Документирование:** Визуальная документация процессов

2. Принципы хорошей схемы:

- **Простота:** Минимум элементов, максимум смысла
- **Ясность:** Четкая иерархия и поток информации
- **Единообразие:** Одинаковые элементы выглядят одинаково
- **Акценты:** Важные элементы выделяются визуально

3. Восприятие цвета и формы:

- **Цвет:** Для категорий, а не украшения
- **Форма:** Для типов элементов (прямоугольник - процесс, ромб - решение)
- **Размер:** Для обозначения важности или объема

Часть 2: Типы схем и их применение (≈1 час)
Задача: Изучить основные типы схем и сферы их применения.
1. Блок-схемы (Flowcharts):
text
[Начало] → [Процесс 1] → [Решение] → (Да) → [Процесс 2] → [Конец]
↓ (Нет)
→ [Альтернативный процесс] → [Конец]

- **Назначение:** Визуализация процессов, алгоритмов
- **Элементы:** Прямоугольники (процессы), ромбы (решения), овалы (начало/конец)

2. Организационные диаграммы:
[Директор]
|

| | |
[Отдел А] [Отдел В] [Отдел С]
| | |
[Сотрудник] [Сотрудник] [Сотрудник]

- **Назначение:** Структура организаций, иерархии

3. Сетевые диаграммы:

- **Назначение:** Топология сетей, связи между компонентами

4. Ментальные карты (Mind Maps):
text
[Центральная идея]
/ | \
[Ветвь 1] [Ветвь 2] [Ветвь 3]
| | |
[Подветвь] [Подветвь] [Подветвь]

- **Назначение:** Структурирование идей, мозговой штурм

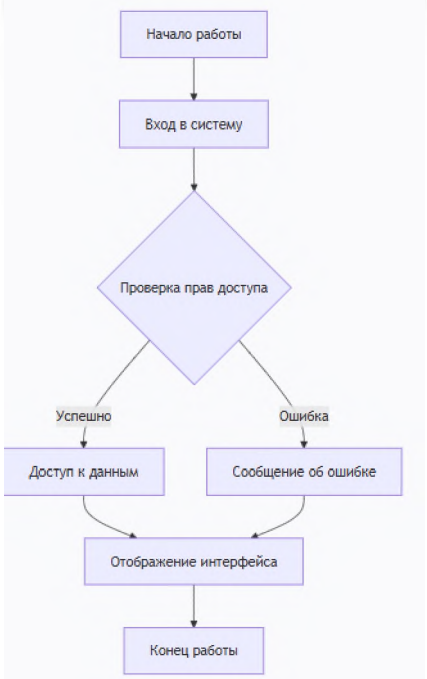
5. Практическое задание "Классификация":

- Студентам даются описания различных систем
- Задача: определить подходящий тип схемы для каждой

Часть 3: Создание блок-схем в Draw.io (≈2 часа)
Задача: Освоить практические навыки создания блок-схем.
1. Знакомство с интерфейсом Draw.io:

- Панель элементов (Basic, Flowchart, Arrows)
- Рабочая область
- Панель форматирования
- Слои и страницы

2. Создание простой блок-схемы процесса:



3. Практическое задание "Процесс авторизации":

- Создать блок-схему процесса входа пользователя в систему
- Использовать стандартные элементы блок-схем
- Применить различные стили соединительных линий

4. Группировка и выравнивание:

- Использование контейнеров для логических групп
- Автоматическое выравнивание элементов
- Распределение с равными интервалами

Часть 4: Создание организационных диаграмм (≈1 час)
Задача: Научиться визуализировать иерархические структуры.
1. Построение структуры компании:
[Генеральный директор]
|
[Технический директор] —— [Финансовый директор]
| |
[Отдел разработки] [Бухгалтерия]
| |
[Старший разработчик] [Главный бухгалтер]
|
[Разработчик]
2. Практическое задание "Оргструктура":

- Создать организационную диаграмму для IT-компании
- Использовать различные формы для разных должностей
- Добавить контактную информацию в tooltips

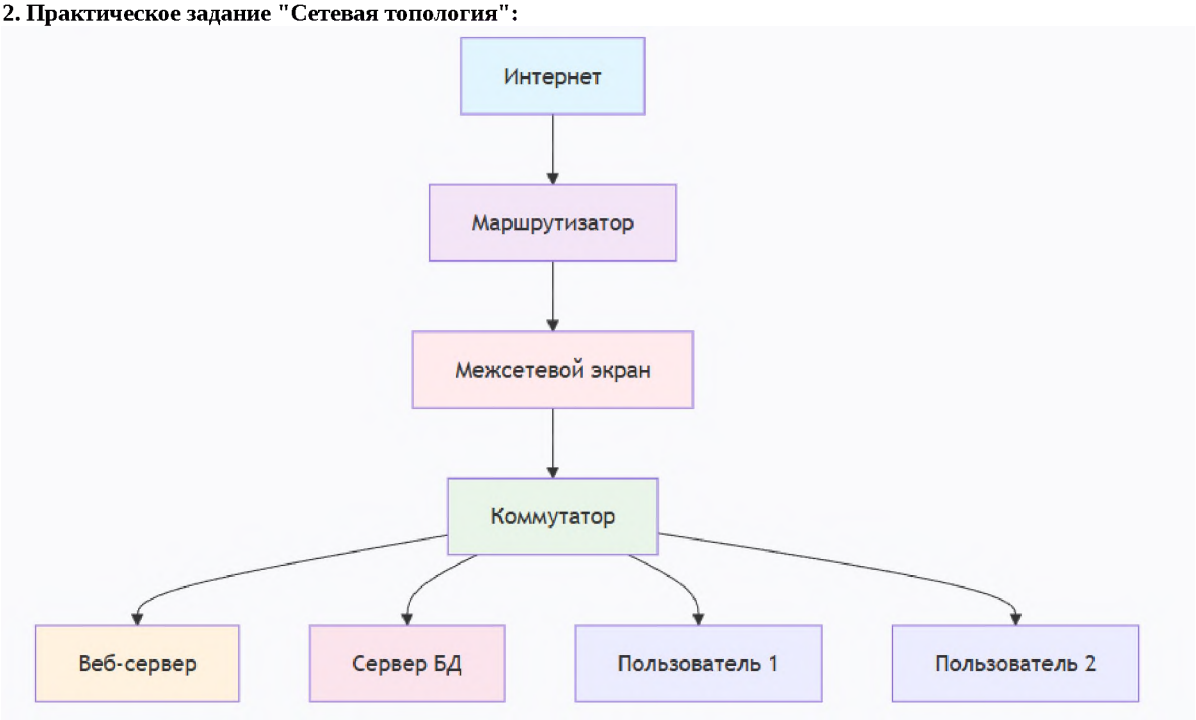
3. Стилизация организационных диаграмм:

- Цветовое кодирование отделов
- Использование иконок для визуального обозначения ролей
- Размер элементов в соответствии с уровнем иерархии

Часть 5: Сетевые диаграммы и инфраструктура (≈1 час 30 минут)
Задача: Освоить создание технических схем сетевой инфраструктуры.
1. Элементы сетевых диаграмм:

- Маршрутизаторы, коммутаторы, серверы
- Межсетевые экраны, точки доступа
- Соединения: Ethernet, Wi-Fi, VPN

2. Практическое задание "Сетевая топология":



3. Обозначения и стандарты:

- Цветовая кодировка для типов устройств
- Обозначения для различных типов соединений
- Легенда и условные обозначения

Часть 6: Ментальные карты и диаграммы связей (≈1 час)
Задача: Научиться визуализировать идеи и взаимосвязи.
1. Создание ментальной карты проекта:
text

[Основной проект]
│
├── [Планирование] → [Исследование] → [Прототип]
├── [Разработка] → [Фронтенд] → [Бэкенд] → [База данных]
├── [Тестирование] → [Юнит-тесты] → [Интеграционные тесты]
└── [Внедрение] → [Деплой] → [Мониторинг]

2. Практическое задание "План разработки":

- Создать ментальную карту для учебного проекта
- Организовать задачи по фазам и приоритетам
- Использовать цветовое кодирование для статусов задач

3. Диаграммы связей для сложных систем:

- Визуализация зависимостей между компонентами
- Выявление узких мест и точек отказа
- Анализ влияния изменений

Часть 7: Профессиональное оформление и экспорт (≈45 минут)

Задача: Научиться доводить схемы до профессионального уровня.

1. Единый стиль оформления:

- Создание и использование шаблонов
- Единая цветовая палитра
- Согласованные шрифты и размеры

2. Добавление легенды и пояснений:

- Создание понятной легенды
- Добавление пояснительных записок
- Нумерация элементов для ссылок в тексте

3. Экспорт в различные форматы:

- **PNG:** Для презентаций и веб-страниц
- **PDF:** Для документации и печати
- **SVG:** Для дальнейшего редактирования
- **XML:** Для совместной работы

4. Практическое задание "Финальная доработка":

- Применить единый стиль к созданным схемам
- Добавить легенды и пояснения
- Экспортировать в различные форматы

Итоговое задание для самостоятельного выполнения:

Сценарий: Вы - системный архитектор, которому нужно подготовить комплект схем для нового проекта.

Задание:

1. **Блок-схема процесса:** Описать процесс обработки заказа в интернет-магазине
2. **Организационная диаграмма:** Представить структуру команды разработки
3. **Сетевая диаграмма:** Изобразить топологию сети офиса
4. **Ментальная карта:** Визуализировать план разработки мобильного приложения

Требования к каждой схеме:

- Четкая визуальная иерархия
- Единый стиль оформления
- Пояснительная легенда
- Профессиональное цветовое решение
- Оптимальный размер и читаемость

Критерии оценки:

- **Информативность (30%):** Полнота и точность представления информации
- **Визуальная ясность (25%):** Простота восприятия и понимания
- **Профессионализм оформления (25%):** Единый стиль, качество визуальных элементов
- **Соответствие стандартам (20%):** Правильное использование условных обозначений

Пример структуры проекта:

text
schemas-project/
├── order-process.drawio
├── team-structure.drawio
├── network-topology.drawio
├── development-plan.drawio
├── exports/
│ ├── order-process.pdf
│ ├── team-structure.png
│ └── network-topology.svg
└── README.md

Это занятие даст студентам практические навыки создания профессиональных схем, которые являются essential инструментом коммуникации в технической среде.

Практическое занятие 29-30: Подготовка графических схем в офисных пакетах

Цель занятия: Освоить создание профессиональных графических схем с использованием встроенных инструментов Microsoft Office и LibreOffice. Научиться эффективно применять SmartArt, фигуры и диаграммы для визуализации информации.

Программное обеспечение:

- Microsoft Office (Word, PowerPoint, Visio - если доступен)
- LibreOffice (Writer, Impress, Draw)
- Google Workspace (Docs, Slides - как альтернатива)

Необходимые знания: Базовые навыки работы с офисными пакетами, понимание принципов создания схем из предыдущих занятий.

План занятия и Ход работы

Часть 1: Введение в инструментарий офисных пакетов (≈45 минут)

Задача: Ознакомиться с возможностями создания схем в различных офисных приложениях.

1. Сравнительный анализ инструментов:

Приложение	Основные возможности	Преимущества	Недостатки
MS Word	SmartArt, фигуры, диаграммы	Интеграция с текстом, простота	Ограниченная функциональность
MS PowerPoint	SmartArt, фигуры, анимация	Лучшая визуализация, слайды	Меньше места для сложных схем
LibreOffice Draw	Профессиональные инструменты	Бесплатность, мощные функции	Сложнее в освоении
Google Slides	Простые фигуры, совместная работа	Облачный, совместный доступ	Ограниченные возможности

2. Обзор панелей инструментов:

- **Вставка** → **Фигуры** (базовые элементы)
- **Вставка** → **SmartArt** (готовые шаблоны)
- **Вставка** → **Диаграммы** (графики и схемы)
- **Формат** (настройка внешнего вида)

3. Практическое задание "Знакомство с интерфейсом":

- Запустить MS Word/PowerPoint и LibreOffice Draw
- Найти и опробовать основные инструменты для создания схем
- Создать простую блок-схему из 3-4 элементов

Часть 2: Создание блок-схем в MS Word (≈1 час 30 минут)

Задача: Освоить построение сложных блок-схем с использованием фигур.

1. Пошаговое создание блок-схемы процесса:



2. Практическое задание "Процесс обработки заявки":

- Создать блок-схему обработки заявки в техподдержку:

[Получение заявки] → [Анализ проблемы] → {Тип проблемы?}
↓ ↓
[Простая] [Сложная]
↓ ↓
[Решение] → [Закрытие] [Эскалация] → [Решение] → [Закрытие]

3. Профессиональные приемы:

- **Использование соединителей:** Прямые, изогнутые, ступенчатые

- **Выравнивание и распределение:** Автоматическое выравнивание элементов
- **Группировка:** Объединение связанных элементов
- **Стили фигур:** Заливка, контуры, тени, эффекты

4. Пример кода стилевого оформления:

```
/* Единый стиль для элементов схемы */
.фигура-процесс {
   заливка: #e1f5fe;
   контур: #01579b;
   тень: 2px 2px 4px rgba(0,0,0,0.2);
   скругление-углов: 5px;
}

.фигура-решение {
   заливка: #f3e5f5;
   контур: #4a148c;
   форма: ромб;
}
```

Часть 3: Использование SmartArt для быстрого создания схем (≈1 час)

Задача: Научиться эффективно использовать готовые шаблоны SmartArt.

1. Типы SmartArt и их применение:

- **Процесс:** Линейные процессы и workflows
- **Иерархия:** Организационные схемы и структуры
- **Цикл:** Циклические процессы
- **Взаимосвязь:** Схемы связей и зависимостей
- **Матрица:** Сравнительные диаграммы

2. Практическое задание "Организационная структура":

- Создать организационную схему компании используя SmartArt "Иерархия":

```
[Генеральный директор]
  ├── [Технический отдел]
  │   ├── [Разработка]
  │   └── [Тестирование]
  ├── [Маркетинг]
  └── [Финансы]
```

3. Настройка и кастомизация SmartArt:

- Добавление/удаление элементов
- Изменение цветовых схем
- Настройка стилей и эффектов
- Преобразование SmartArt в фигуры для тонкой настройки

4. Работа с текстом в SmartArt:

- Форматирование текста
- Использование маркированных списков
- Настройка шрифтов и размеров

Часть 4: Создание схем в MS PowerPoint (≈1 час 30 минут)

Задача: Использовать преимущества PowerPoint для создания презентабельных схем.

1. Особенности работы в PowerPoint:

- Большее рабочее пространство
- Возможности анимации элементов
- Использование мастер-слайдов для единообразия
- Экспорт отдельных слайдов как изображений

2. Практическое задание "Интерактивная схема":

- Создать схему архитектуры веб-приложения с анимацией появления элементов:

```
[Пользователь] → [Веб-интерфейс] → [Сервер приложения] → [База данных]
      ↓                ↓                  ↓                  ↓
   [Браузер]      [HTML/CSS/JS]      [Java/Python]        [MySQL]
```

3. Приемы профессионального дизайна:

- **Цветовые палитры:** Использование корпоративных цветов
- **Типографика:** Согласованные шрифты и размеры
- **Выравнивание:** Сетки и направляющие
- **Единый стиль:** Использование тем оформления

Часть 5: Работа в LibreOffice Draw (≈1 час)

Задача: Освоить бесплатные альтернативы для создания схем.

1. Интерфейс LibreOffice Draw:

- Панель рисования
- Галерея элементов
- Стили и форматирование
- Слои и группировка

2. Практическое задание "Сетевая диаграмма":

- Создать схему сетевой инфраструктуры офиса:

```
[Интернет] → [Маршрутизатор] → [Коммутатор] → [Рабочие станции]
                ↘ [Сервер] → [Принтер]
```

3. Сравнение с MS Office:

- Аналогичные возможности
- Другие названия инструментов
- Особенности экспорта и сохранения

Часть 6: Интеграция схем в документы (≈45 минут)

Задача: Научиться эффективно встраивать схемы в различные типы документов.

1. Оптимальные форматы вставки:

- **Встроенный объект:** Для дальнейшего редактирования
- **Связь с файлом:** Для автоматического обновления
- **Изображение:** Для финальной версии документа

2. Практическое задание "Комплексный документ":

- Создать техническое задание со встроенными схемами:
  - Титульная страница
  - Блок-схема процесса
  - Организационная схема
  - Схема архитектуры системы

3. Подписи и нумерация:

- Добавление автоматических подписей
- Создание перекрестных ссылок
- Генерация списка иллюстраций

Часть 7: Экспорт и совместная работа (≈30 минут)

Задача: Освоить методы экспорта схем и организации совместной работы.

1. Форматы экспорта:

- **PDF:** Для печати и распространения
- **PNG/JPEG:** Для веб-страниц и презентаций
- **SVG:** Для дальнейшего редактирования в векторных редакторах

2. Совместная работа:

- Комментарии и рецензирование
- Сравнение версий
- Защита от изменений

3. Практическое задание "Финальная подготовка":

- Экспортировать созданные схемы в различные форматы
- Настроить параметры качества и размера
- Проверить читаемость в разных форматах

Итоговое задание для самостоятельного выполнения:

Сценарий: Вы - технический писатель, которому нужно подготовить комплект схем для руководства пользователя.

Задание:

1. **В MS Word:** Создать блок-схему процесса установки программного обеспечения
2. **В MS PowerPoint:** Разработать организационную схему отделов компании
3. **В LibreOffice Draw:** Построить схему сетевой инфраструктуры
4. **Интеграция:** Объединить все схемы в едином документе с автоматическими подписями

Требования к каждой схеме:

- Профессиональное оформление в едином стиле
- Четкая визуальная иерархия
- Подписи и пояснения
- Оптимальный размер и читаемость

Критерии оценки:

- **Техническая корректность (30%):** Правильное использование элементов схем
- **Визуальное качество (25%):** Профессиональный дизайн, единый стиль
- **Функциональность (25%):** Четкость восприятия, информативность
- **Интеграция (20%):** Качество встраивания в документы, правильные подписи

Пример структуры проекта:

```
office-schemas/
├── installation-process.docx
├── organization-structure.pptx
├── network-infrastructure.odg
├── final-document/
│   ├── technical-manual.docx
│   └── export/
│       ├── process.pdf
│       ├── structure.png
│       └── network.svg
└── README.md
```

Это занятие даст студентам практические навыки создания качественных схем с использованием доступных офисных инструментов, что является востребованной компетенцией в большинстве организаций.

Практическое занятие 31-34: Подготовка графических схем в специализированных визуальных редакторах

Цель занятия: Освоить профессиональные инструменты для создания сложных графических схем, изучить специализированные нотации и научиться выбирать оптимальный инструмент для различных задач.

Программное обеспечение:

- **Диаграммы и блок-схемы:** Draw.io (Diagrams.net), Lucidchart, Visio
- **UML-диаграммы:** PlantUML, Visual Paradigm
- **Ментальные карты:** XMind, MindManager
- **Сетевые диаграммы:** yEd Graph Editor, Cisco Network Diagrams
- **Универсальные:** Miro, Figma

Необходимые знания: Базовые навыки создания схем в офисных пакетах, понимание принципов визуализации информации.

План занятия и Ход работы

Часть 1: Обзор специализированных редакторов (≈1 час)

Задача: Познакомиться с ландшафтом профессиональных инструментов.

1. Классификация инструментов:

Категория	Основные инструменты
Бизнес-диаграммы	Draw.io, Lucidchart, Visio
UML-моделирование	PlantUML, Visual Paradigm

Категория	Основные инструменты
Ментальные карты	XMind, MindManager
Сетевые диаграммы	yEd, Cisco Packet Tracer
Интерактивные доски	Miro, FigJam

2. Критерии выбора инструмента:

- Сложность создаваемых схем
- Требования к сотрудничеству
- Необходимость интеграции с другими системами
- Бюджет и лицензирование

3. Практическое задание "Анализ инструментов":

- Сравнить 2-3 инструмента по критериям:
  - Удобство интерфейса
  - Богатство библиотек элементов
  - Возможности экспорта
  - Стоимость и лицензия

Часть 2: Глубокое погружение в Draw.io/Diagrams.net (≈2 часа)

Задача: Освоить наиболее популярный бесплатный инструмент для создания диаграмм.

1. Интерфейс и основные возможности:

- Рабочее пространство и панели инструментов
- Библиотеки элементов (Basic, Advanced, AWS, Azure и др.)
- Слои и страницы
- Горячие клавиши и продуктивность

2. Создание комплексной блок-схемы бизнес-процесса:

3. Практическое задание "Процесс электронной коммерции":

- Создать детальную схему процесса заказа товара:

```
text
[Поиск товара] → [Добавление в корзину] → [Оформление заказа] → {Наличие товара?}
   ↓                                  ↓
[Уведомление] ← [Отмена заказа]    [Подтверждение] → [Доставка] → [Завершение]
```

4. Профессиональные приемы в Draw.io:

- **Стили и форматы:** Глобальное изменение стилей
- **Шаблоны и библиотеки:** Создание собственных элементов
- **Автоматическое выравнивание:** Использование сеток и направляющих
- **Экспорт:** Настройка качества и форматов

Часть 3: UML-диаграммы в PlantUML (≈1 час 30 минут)

Задача: Освоить текстовый подход к созданию диаграмм.

1. Преимущества текстовых диаграмм:

- Версионный контроль (Git)
- Легкое изменение и повторное использование
- Автоматизация генерации
- Независимость от графического интерфейса

2. Создание диаграммы последовательности:

```
plantuml
@startuml
actor Пользователь
participant "Веб-интерфейс" as UI
participant "Сервер" as Server
participant "База данных" as DB

Пользователь -> UI: Запрос данных
UI -> Server: API вызов
Server -> DB: SQL запрос
DB --> Server: Результат
Server --> UI: JSON ответ
UI --> Пользователь: Отображение данных
@enduml
```

3. Практическое задание "Диаграмма классов":

- Создать UML-диаграмму для системы управления библиотекой:

```
plantuml
@startuml
class Book {
  - title: String
  - author: String
  - isbn: String
```

```
  + borrow()
  + return()
}

class User {
  - name: String
  - userId: String
  + searchBook()
  + reserveBook()
}

class Library {
  - books: List<Book>
  - users: List<User>
  + addBook()
  + removeBook()
}

User "1" -- "*" Book : borrows
Library "1" -- "*" Book : contains
Library "1" -- "*" User : manages
@enduml
```

4. Интеграция PlantUML с другими инструментами:

- Плагины для VS Code, IntelliJ IDEA
- Использование в документации (Markdown, Confluence)
- Автоматическая генерация из кода

Часть 4: Создание ментальных карт в XMind (≈1 час)

Задача: Освоить технику визуализации идей и планирования.

1. Структура ментальной карты:

```
text
[Центральная тема]
   |
   ├── [Основная ветвь 1]
   │    ├── [Подветвь 1.1]
   │    └── [Подветвь 1.2]
   ├── [Основная ветвь 2]
   │    ├── [Подветвь 2.1]
   │    └── [Подветвь 2.2]
   └── [Основная ветвь 3]
```

2. Практическое задание "План проекта":

- Создать ментальную карту для разработки мобильного приложения:

```
text
[Мобильное приложение]
   |
   ├── [Планирование] → [Исследование рынка] → [Прототип]
   ├── [Дизайн] → [UI/UX] → [Графика] → [Анимации]
   ├── [Разработка] → [Фронтенд] → [Бэкенд] → [API]
   ├── [Тестирование] → [Юнит-тесты] → [Интеграция]
   └── [Запуск] → [Публикация] → [Маркетинг]
```

3. Расширенные возможности XMind:

- Добавление связей между ветвями
- Использование маркеров и меток
- Вставка изображений и заметок
- Экспорт в различные форматы

Часть 5: Сетевые диаграммы в yEd Graph Editor (≈1 час 30 минут)

Задача: Научиться создавать профессиональные сетевые схемы.

1. Особенности yEd:

- Мощные алгоритмы автоматического размещения
- Богатые библиотеки сетевых элементов
- Поддержка различных стандартов оформления

2. Практическое задание "Сетевая инфраструктура офиса":

3. Автоматическое размещение в yEd:

- Органические, иерархические, радиальные схемы
- Настройка параметров размещения
- Ручная корректировка автоматических расположений

Часть 6: Совместная работа в Miro (≈1 час)

Задача: Освоить инструменты для командной работы над схемами.

1. Возможности Miro для совместной работы:

- Реальное время collaboration
- Комментарии и обсуждения
- История изменений
- Интеграция с другими инструментами

2. Практическое задание "Командное проектирование":

- Создать схему архитектуры в команде из 2-3 человек
- Использовать комментарии для обсуждения решений
- Применить шаблоны для ускорения работы

3. Расширенные функции Miro:

- Бесконечные доски
- Smart Drawing (распознавание фигур)
- Презентационный режим
- Экспорт и совместный доступ

Часть 7: Оптимизация workflow и интеграция (≈1 час)
Задача: Научиться эффективно комбинировать инструменты.
1. Выбор инструмента по задаче:

Задача	Рекомендуемый инструмент
Быстрая блок-схема	Draw.io
Документирование архитектуры	PlantUML
Планирование проекта	XMind
Сетевая инфраструктура	yEd
Командная работа	Miro

2. Интеграция инструментов:

- Экспорт из Draw.io в SVG для доработки в Figma
- Использование PlantUML в Markdown-документации
- Импорт ментальных карт в инструменты управления проектами

3. Практическое задание "Комбинированный подход":

- Создать ментальную карту в XMind
- Детализировать ключевые процессы в Draw.io
- Оформить архитектурные решения в PlantUML
- Объединить все в презентацию

Итоговое задание для самостоятельного выполнения:
Сценарий: Вы - архитектор программного обеспечения, которому нужно подготовить полный комплект документации для нового проекта.
Задание:

1. **В Draw.io:** Создать детальную блок-схему бизнес-процесса заказа
2. **В PlantUML:** Разработать UML-диаграммы (классов, последовательности)
3. **В XMind:** Построить ментальную карту плана разработки
4. **В yEd:** Спроектировать схему развертывания инфраструктуры
5. **Интеграция:** Объединить все схемы в связную документацию

Требования к проекту:

- Единый стиль оформления всех схем
- Согласованность между различными диаграммами
- Профессиональное качество визуализации
- Полнота и точность отражения системы

Критерии оценки:

- **Техническая корректность (30%):** Правильное использование нотаций и элементов
- **Визуальное качество (25%):** Профессиональный дизайн, единый стиль
- **Информативность (25%):** Полнота и ясность представления информации
- **Интеграция (20%):** Согласованность между различными схемами проекта

Пример структуры проекта:

```
text
professional-schemas/
├── business-process/
│   └── order-process.drawio
├── architecture/
│   ├── class-diagram.puml
│   └── sequence-diagram.puml
├── planning/
│   └── project-plan.xmind
├── infrastructure/
│   └── deployment-diagram.graphml
└── documentation/
    └── technical-specification.md
```

└── README.md
Это занятие даст студентам профессиональные навыки работы с промышленными инструментами визуализации, что является критически важной компетенцией для архитекторов, аналитиков и разработчиков.
Практическое занятие 35 - 38. Подготовка графических схем с помощью программ-генераторов, поддерживающих языки разметки, предназначенные для описания графических схем
Отлично! Это продвинутое занятие посвящено программному подходу к созданию схем. Разработаем комплексный план на 4 академических часа.

Практическое занятие 35-38: Подготовка графических схем с помощью программ-генераторов на языках разметки
Цель занятия: Освоить декларативный подход к созданию схем с использованием языков разметки. Научиться генерировать сложные диаграммы из текстовых описаний, интегрировать их в документацию и автоматизировать процесс создания.
Программное обеспечение:

- **Диаграммы:** Graphviz, Mermaid.js, PlantUML
- **Визуализация данных:** D3.js, Vega-Lite
- **Редакторы кода:** VS Code с плагинами
- **Системы контроля версий:** Git
- **Сборка документации:** MkDocs, Sphinx, Jupyter Notebook

Необходимые знания: Базовое понимание синтаксиса разметки, опыт работы с текстовыми редакторами.

План занятия и Ход работы
Часть 1: Философия декларативного подхода (≈1 час)
Задача: Понять преимущества текстового описания схем перед визуальным редактированием.
1. Преимущества языков разметки для диаграмм:

- **Версионный контроль:** Диаграммы как код (Diagrams as Code)
- **Повторное использование:** Шаблоны и библиотеки компонентов
- **Автоматизация:** Генерация из данных и скриптов
- **Совместная работа:** Конфликт-фри мерджи в Git
- **Доступность:** Чтение скринридерами

2. Сравнительный анализ инструментов:

Инструмент	Язык	Лучшее применение	Преимущества
Graphviz	DOT	Графы, сети	Мощные алгоритмы размещения
Mermaid	Markdown-like	Документация	Простота, веб-интеграция
PlantUML	UML-нотация	Архитектура ПО	Богатые UML-диаграммы
D3.js	JavaScript	Интерактивность	Полный контроль, анимация

3. Практическое задание "Hello World":

- Создать простейшую диаграмму в каждом инструменте
- Сравнить синтаксис и подходы

Часть 2: Graphviz и язык DOT (≈2 часа)
Задача: Освоить мощный инструмент для автоматического размещения графов.
1. Базовый синтаксис DOT:
dot
```
digraph ПростойГраф {
    rankdir=TB;  // Направление графа (TB - сверху вниз)
    node [shape=rectangle, style=filled, color=lightblue];

    A [label="Начало"];
    B [label="Процесс 1"];
    C [label="Процесс 2"];
    D [label="Конец", shape=oval];

    A -> B -> C -> D;
    B -> D [label="Альтернативный путь"];
}
```
2. Создание организационной диаграммы:
dot
```
digraph Организация {
    rankdir=TB;
    node [shape=box, style=filled];

    CEO [label="Генеральный директор", color="#4CAF50"];
```

```
    subgraph cluster_tech {
      label="Технический отдел";
      color=lightgray;
      CTO [label="Технический директор", color="#2196F3"];
      Dev [label="Отдел разработки", color="#2196F3"];
      QA [label="Отдел тестирования", color="#2196F3"];
    }

    subgraph cluster_marketing {
      label="Маркетинг";
      color=lightgray;
      CMO [label="Директор по маркетингу", color="#FF9800"];
      Adv [label="Рекламный отдел", color="#FF9800"];
    }

    CEO -> CTO;
    CEO -> CMO;
    CTO -> Dev;
    CTO -> QA;
    CMO -> Adv;
}
```

3. Практическое задание "Сетевая топология":

- Создать схему сети офиса с использованием различных форм и цветов

4. Автоматическое размещение:

- Алгоритмы размещения (dot, neato, fdp, sfdp)
- Настройка рангов и кластеров
- Управление направлением и пропорциями

Часть 3: Mermaid.js - диаграммы для документации (≈2 часа)
Задача: Освоить простой и мощный инструмент для веб-документации.
1. Блок-схемы в Mermaid:
2. Последовательности в Mermaid:
3. Практическое задание "Процесс заказа":

- Создать полную схему процесса заказа в интернет-магазине
- Использовать различные типы узлов и стилизацию

4. Интеграция Mermaid:

- Встраивание в Markdown-документы
- Использование в GitHub/GitLab
- Интеграция с веб-приложениями

Часть 4: PlantUML для архитектурных диаграмм (≈1 час 30 минут)
Задача: Освоить профессиональный инструмент для UML и архитектурных диаграмм.
1. Диаграмма компонентов:

```
plantuml
@startuml
!define RECTANGLE class
skinparam componentStyle rectangle

package "Веб-приложение" {
  [Веб-интерфейс] as UI
  [API Gateway] as API
  [Сервис аутентификации] as Auth
  [Сервис заказов] as Orders
}

package "Внешние сервисы" {
  [Платежная система] as Payment
  [СМС-шлюз] as SMS
}

database "База данных" as DB

UI --> API
API --> Auth
API --> Orders
Orders --> DB
Orders --> Payment
Auth --> SMS

note right of UI: Angular SPA
note right of API: Spring Boot
note right of DB: PostgreSQL
@enduml
```

2. Диаграмма развертывания:

```
plantuml
```

```
@startuml
node "Пользователи" as Users

cloud "Интернет" as Internet

node "Облачный провайдер" {
   node "Веб-сервер" as Web {
      component "Nginx" as Nginx
   }

   node "Сервер приложения" as App {
      component "Docker" as Docker
      component "Spring Boot" as Spring
   }

   node "База данных" as DB {
      component "PostgreSQL" as PostgreSQL
   }
}

Users --> Internet
Internet --> Nginx
Nginx --> Spring
Spring --> PostgreSQL

note right of Docker: Контейнеризация
note left of PostgreSQL: Репликация\nи бэкапы
@enduml
```

3. Практическое задание "Архитектура микросервисов":

- Создать диаграмму взаимодействия микросервисов
- Использовать различные стили и нотации

Часть 5: D3.js для интерактивных диаграмм (≈1 час)
Задача: Познакомиться с созданием интерактивных и анимированных схем.
1. Базовый пример графа:

```
javascript
// HTML: <div id="graph"></div>
const data = {
  nodes: [
    {id: "A", group: 1},
    {id: "B", group: 1},
    {id: "C", group: 2},
    {id: "D", group: 2}
  ],
  links: [
    {source: "A", target: "B"},
    {source: "B", target: "C"},
    {source: "C", target: "D"},
    {source: "A", target: "D"}
  ]
};

// Создание силы-directed графа
const simulation = d3.forceSimulation(data.nodes)
  .force("link", d3.forceLink(data.links).id(d => d.id))
  .force("charge", d3.forceManyBody())
  .force("center", d3.forceCenter(300, 150));
```

2. Практическое задание "Интерактивный органайзер":

- Создать простой интерактивный граф с возможностью перетаскивания узлов

Часть 6: Интеграция и автоматизация (≈1 час 30 минут)
Задача: Научиться интегрировать генераторы диаграмм в рабочие процессы.
1. Интеграция с системами документации:

```
markdown
# Техническая документация

## Архитектура системы

```mermaid
flowchart TD
   A[Клиент] --> B[Балансировщик]
   B --> C[Сервер 1]
   B --> D[Сервер 2]
```

Схема базы данных

```
plantuml
@startuml
```
```

```
entity "Пользователи" {
  * id : integer
  --
  * username : varchar
  * email : varchar
}
@enduml
text
```

2. Автоматическая генерация из данных:

```python
# Генератор диаграмм из CSV
import pandas as pd
import subprocess

def generate_org_chart(csv_file, output_file):
    df = pd.read_csv(csv_file)

    dot_code = 'digraph Организация {\nrankdir=TB;\nnode [shape=box];\n'

    for _, row in df.iterrows():
        dot_code += f'"{row["name"]}" [label="{row["name"]}\\n{row["position"]}"];\n'
        if pd.notna(row["manager"]):
            dot_code += f'"{row["manager"]}" -> "{row["name"]}";\n'

    dot_code += '}'

    with open('temp.dot', 'w', encoding='utf-8') as f:
        f.write(dot_code)

    subprocess.run(['dot', '-Tpng', 'temp.dot', '-o', output_file])
```

3. CI/CD для диаграмм:

```yaml
# .gitlab-ci.yml
generate_diagrams:
  image: python:3.9
  script:
    - apt-get update && apt-get install -y graphviz
    - pip install plantuml
    - python generate_diagrams.py
    - plantuml docs/**/*.puml
  artifacts:
    paths:
      - docs/diagrams/
```

4. Практическое задание "Автоматизированный пайплайн":

- Создать скрипт для генерации набора диаграмм из структурированных данных
- Настроить автоматическое обновление диаграмм при изменении данных

Итоговое задание для самостоятельного выполнения:

Сценарий: Вы - технический писатель в IT-компании, которому нужно создать полный комплект автоматически генерируемой документации.

Задание:

1. **Graphviz:** Создать схему организационной структуры из CSV-файла
2. **Mermaid:** Разработать блок-схемы бизнес-процессов в Markdown-документации
3. **PlantUML:** Построить архитектурные диаграммы системы
4. **Интеграция:** Объединить все диаграммы в единую документацию
5. **Автоматизация:** Настроить скрипт для автоматической генерации при изменении данных

Требования к проекту:

- Все диаграммы генерируются из текстовых описаний
- Единый стиль оформления
- Возможность автоматического обновления
- Интеграция с системой контроля версий

Критерии оценки:

- **Техническая корректность (30%):** Правильное использование синтаксиса и нотаций
- **Автоматизация (25%):** Эффективность скриптов генерации
- **Качество визуализации (25%):** Читаемость и профессиональный вид диаграмм
- **Интеграция (20%):** Согласованность между диаграммами и документацией

Пример структуры проекта:

```
diagrams-as-code/
├── src/
│   ├── data/
│   │   ├── organization.csv
│   │   └── processes.json
│   ├── scripts/
│   │   ├── generate_org_chart.py
```

```
│   │   └── generate_architecture.py
│   └── diagrams/
│       ├── org_chart.dot
│       ├── processes.mmd
│       └── architecture.puml
├── docs/
│   ├── index.md
│   ├── organization.md
│   └── architecture.md
├── output/
│   ├── org_chart.png
│   ├── processes.svg
│   └── architecture.pdf
└── .github/
    └── workflows/
        └── generate-diagrams.yml
```

Это занятие даст студентам продвинутые навыки работы с промышленными инструментами автоматической генерации диаграмм, что является ключевой компетенцией в современных DevOps и документационных процессах.
Практическое занятие 39 - 42. Преобразование файлов гра-фических схем из векторных форматов в растровые форматы Практическое занятие 43 - 46. Отображение алгоритма или процесса с помощью блок-схемы, логическую структуру ба-зы данных с помощью диаграммы "сущность - связь" Практическое занятие 47 - 50. Отображение структуры ком-пьютерной сети или системы с помощью графической схе-мы Практическое занятие 51 - 54. Получение иллюстративные снимки экрана. Практическое занятие 55 - 58. Оптимизация снимка экрана для размещения на странице заданного формата
Отлично! Объединим эти темы в комплексный практикум по визуализации и оптимизации графики. Разработаем интегрированный план на 20 академических часов.

Практический цикл: Профессиональная работа с графикой и схемами (Занятия 39-58)
Общая цель цикла: Освоить полный цикл работы с графикой - от создания векторных схем до оптимизации растровых изображений для публикации.

Модуль 1: Преобразование векторных форматов в растровые (Занятия 39-42)
Задача: Освоить конвертацию векторных схем в растровые форматы с сохранением качества.
Часть 1: Теория преобразования форматов (≈1 час)
Часть 2: Практикум конвертации (≈3 часа)
Инструменты: Inkscape, GIMP, командные утилиты
1. Пакетная обработка в Inkscape:
bash
```
# Конвертация SVG в PNG с заданным DPI
inkscape input.svg --export-filename=output.png --export-dpi=300

# Пакетная обработка
for file in *.svg; do
    inkscape "$file" --export-filename="${file%.svg}.png" --export-dpi=150
done
```

2. Практическое задание "Конвейер конвертации":

- Создать скрипт для пакетной конвертации схем
- Настроить разные параметры для веба (72 DPI) и печати (300 DPI)
- Сравнить качество PNG vs JPEG для схем с текстом

Модуль 2: Создание профессиональных схем (Занятия 43-50)
Блок-схемы алгоритмов (Занятия 43-46)
Пример сложного алгоритма:
ER-диаграммы баз данных (Занятия 45-46)
Схемы компьютерных сетей (Занятия 47-50)

Модуль 3: Работа со снимками экрана (Занятия 51-58)
Создание профессиональных скриншотов (Занятия 51-54)
Инструменты: ShareX (Windows), Shutter (Linux), CleanShot (Mac)
1. Настройка горячих клавиш:
yaml
```
# Конфигурация ShareX
hotkeys:
  fullscreen: Ctrl+Shift+1
  region: Ctrl+Shift+2
  active_window: Ctrl+Shift+3
  scrolling_capture: Ctrl+Shift+4

destinations:
  after_capture: [save, copy]
  after_upload: [copy_url]

image_settings:
  format: png
  jpeg_quality: 95
```

2. Практическое задание "Документирование интерфейса":

- Сделать серию скриншотов веб-приложения
- Захватить различные состояния UI
- Создать аннотации для пояснения

Оптимизация скриншотов (Занятия 55-58)

1. Анализ исходного файла:

```
python
from PIL import Image
import os

def analyze_image(image_path):
    with Image.open(image_path) as img:
        print(f"Размер: {img.size}")
        print(f"Формат: {img.format}")
        print(f"Режим: {img.mode}")
        print(f"Размер файла: {os.path.getsize(image_path)} байт")

# Использование
analyze_image("screenshot.png")
```

2. Скрипт пакетной оптимизации:

```
python
from PIL import Image
import os

def optimize_screenshots(input_dir, output_dir, target_size=(800, 600), quality=85):
    """
    Оптимизация скриншотов для веба
    """
    if not os.path.exists(output_dir):
        os.makedirs(output_dir)

    for filename in os.listdir(input_dir):
        if filename.lower().endswith(('.png', '.jpg', '.jpeg')):
            input_path = os.path.join(input_dir, filename)
            output_path = os.path.join(output_dir, filename)

            with Image.open(input_path) as img:
                # Изменение размера с сохранением пропорций
                img.thumbnail(target_size, Image.Resampling.LANCZOS)

                # Конвертация в RGB если нужно
                if img.mode in ('RGBA', 'P'):
                    img = img.convert('RGB')

                # Сохранение с оптимизацией
                img.save(output_path, 'JPEG', quality=quality, optimize=True)

            print(f"Оптимизирован: {filename}")

# Использование
optimize_screenshots('raw_screenshots', 'optimized', (1200, 800), 80)
```

3. Практическое задание "Веб-галерея":

- Создать набор скриншотов приложения
- Оптимизировать для разных устройств
- Построить responsive галерею

Комплексное итоговое задание для всего цикла:

Сценарий: Вы - технический писатель в IT-компании, которому нужно подготовить полный комплект документации для нового продукта.

Задание:

1. Создание схем и диаграмм

- **Блок-схема:** Алгоритм работы системы аутентификации
- **ER-диаграмма:** Структура базы данных пользователей
- **Сетевая диаграмма:** Архитектура развертывания системы

2. Документирование интерфейса

- Серия скриншотов основного workflow приложения
- Аннотированные изображения ключевых функций
- Сравнительные скриншоты "до/после" изменений

3. Оптимизация и публикация

```
bash
# Структура проекта
documentation/
├── src/
│   ├── diagrams/          # Исходные векторные файлы
│   │   ├── auth-flow.svg
│   │   ├── database-er.svg
│   │   └── network-arch.svg
```

```
│   └── screenshots/        # Исходные скриншоты
│       ├── login-process/
│       ├── user-dashboard/
│       └── admin-panel/
├── optimized/              # Оптимизированные файлы
│   ├── diagrams/
│   │   ├── auth-flow.png
│   │   ├── database-er.jpg
│   │   └── network-arch.webp
│   └── screenshots/
│       ├── web/           # Для сайта (72 DPI)
│       ├── print/         # Для печати (300 DPI)
│       └── mobile/        # Для мобильных (150 DPI)
└── scripts/
    ├── convert_diagrams.py
    ├── optimize_screenshots.py
    └── generate_gallery.py
```

4. Автоматизация процесса

```python
# automate_documentation.py
import subprocess
import os
from datetime import datetime

class DocumentationBuilder:
    def __init__(self):
        self.timestamp = datetime.now().strftime("%Y%m%d_%H%M")

    def build_diagrams(self):
        """Конвертация всех диаграмм"""
        diagrams = {
            'svg': ['auth-flow', 'database-er', 'network-arch'],
            'drawio': ['system-overview']
        }

        for format, names in diagrams.items():
            for name in names:
                self.convert_diagram(f"src/diagrams/{name}.{format}")

    def optimize_screenshots(self):
        """Оптимизация скриншотов"""
        sizes = {
            'web': (1200, 800),
            'mobile': (768, 1024),
            'print': (2480, 3508)  # A4 at 300 DPI
        }

        for size_name, dimensions in sizes.items():
            self.process_screenshots(dimensions, size_name)

    def generate_report(self):
        """Генерация отчета о проделанной работе"""
        # Анализ размеров файлов до/после оптимизации
        # Создание сравнительной таблицы
        pass

# Запуск процесса
if __name__ == "__main__":
    builder = DocumentationBuilder()
    builder.build_diagrams()
    builder.optimize_screenshots()
    builder.generate_report()
```

Критерии оценки для всего цикла:

Компонент	Критерии
Качество схем	Правильность нотации, читаемость, пол
Профессионализм скриншотов	Четкость, релевантность, аннотации
Оптимизация	Эффективность сжатия, сохранение каче

Компонент	Критерии
Автоматизация	Скрипты конвертации, пакетная обработ
Документирование	Структура проекта, README, коммента
Интеграция	Согласованность всех компонентов

Этот комплексный подход позволяет студентам освоить полный цикл работы с графикой - от создания профессиональных схем до их оптимизации и интеграции в документацию, что является ключевым навыком для технических специалистов.
Практическое занятие 39 - 42. Преобразование файлов гра-фических схем из векторных форматов в растровые форматы
Отлично! Разработаем детальный план практического занятия по преобразованию векторных графических схем в растровые форматы.

Практическое занятие 39-42: Преобразование файлов графических схем из векторных форматов в растровые форматы
Цель занятия: Освоить профессиональные методы конвертации векторных графических файлов в растровые форматы с сохранением качества и контролем параметров выходных файлов.
Программное обеспечение:

- **Векторные редакторы:** Inkscape, Adobe Illustrator
- **Растровые редакторы:** GIMP, Adobe Photoshop
- **Командные утилиты:** ImageMagick, Ghostscript
- **Онлайн-конвертеры:** CloudConvert, Convertio
- **Пакетные обработчики:** XnConvert, BatchPhoto

Необходимые знания: Базовое понимание различий между векторной и растровой графикой, опыт работы с файлами различных форматов.

План занятия и Ход работы
Часть 1: Теоретические основы преобразования (≈1 час 30 минут)
Задача: Понять принципы преобразования и критерии выбора форматов.
1. Различия векторной и растровой графики:

Параметр	Векторная графика
Основной принцип	Математические формулы
Масштабирование	Без потери качества
Размер файла	Зависит от сложности
Основные форматы	SVG, PDF, EPS, AI
Лучшее применение	Логотипы, схемы, чертежи

2. Ключевые параметры преобразования:

- **Разрешение (DPI):** 72 DPI (веб), 150 DPI (офис), 300 DPI (печать)
- **Размер в пикселях:** Ширина × Высота
- **Цветовой режим:** RGB (веб), CMYK (печать)
- **Сжатие:** С потерями (JPEG), без потерь (PNG)

3. Матрица выбора формата:

Исходный файл	Цель	Рекомендуемый формат	Настройки
Схема с текстом	Веб-сайт	PNG	72 DPI, RGB
Технический чертеж	Печать	TIFF	300 DPI, CMYK

Исходный файл	Цель	Рекомендуемый формат	Настройки
Презентация	Документ Word	JPEG	150 DPI, RGB
Логотип	Мобильное приложение	PNG	Различные размеры

Часть 2: Ручная конвертация в графических редакторах (≈3 часа)
Задача: Освоить интерактивные методы конвертации.
1. Конвертация в Inkscape:
bash
 # Подготовка тестовых файлов
- Создать SVG-схему процесса
- Создать PDF-документ с диаграммами
- Создать AI-файл с логотипом

Пошаговая процедура:

1. Открыть векторный файл в Inkscape
2. **Файл → Сохранить как**
3. Выбрать формат PNG
4. Настроить параметры экспорта:
   - **Разрешение:** 300 DPI для печати, 96 DPI для веба
   - **Размер:** Задать ширину/высоту в пикселях
   - **Область экспорта:** Страница, рисунок, выделение
5. Сохранить

2. Практическое задание "Многоформатный экспорт":

- Исходный файл: technical-diagram.svg
- Создать версии:
  - diagram-web.png (800×600px, 72 DPI)
  - diagram-print.tiff (2480×3508px, 300 DPI)
  - diagram-mobile.jpg (400×300px, 150 DPI)

3. Конвертация в GIMP:
python
 # Алгоритм работы в GIMP
1. Открыть векторный файл (автоматическая растеризация)
2. Проверить разрешение: «Изображение → Размер печати»
3. Настроить размер: «Изображение → Размер изображения»
4. Экспорт: «Файл → Экспортировать как»
5. Выбрать формат и настройки качества

4. Сравнение результатов:

- Создать таблицу сравнения размеров файлов и качества

Формат	Размер файла	Качество при увеличении	Применение
PNG	450 КБ	Отличное	Веб, документы
JPEG	120 КБ	Хорошее	Презентации
TIFF	2.1 МБ	Идеальное	Печать

Часть 3: Пакетная обработка файлов (≈2 часа 30 минут)
Задача: Освоить автоматизацию процесса конвертации.
1. Использование ImageMagick:
bash
 # Установка (Ubuntu/Debian)
sudo apt-get install imagemagick

 # Базовая конвертация SVG в PNG
convert input.svg output.png

 # Конвертация с указанием DPI
convert -density 300 input.svg -resize 50% output.png

 # Пакетная обработка SVG файлов
for file in *.svg; do

```
    convert -density 150 "$file" "${file%.svg}.png"
done

# Создание превью разного размера
convert input.svg -resize 800x600 large.png
convert input.svg -resize 400x300 medium.png
convert input.svg -resize 200x150 small.png
```

2. Скрипт для комплексной обработки:

```
bash
#!/bin/bash
# batch_convert.sh - Пакетная конвертация векторных файлов

INPUT_DIR="vector_files"
OUTPUT_DIR="raster_files"
FORMATS=("png" "jpg" "tiff")

# Создание выходных директорий
for format in "${FORMATS[@]}"; do
    mkdir -p "$OUTPUT_DIR/$format"
done

# Обработка SVG файлов
for svg_file in "$INPUT_DIR"/*.svg; do
    if [ -f "$svg_file" ]; then
        filename=$(basename "$svg_file" .svg)

        # Конвертация в PNG (без потерь)
        convert -density 150 "$svg_file" "$OUTPUT_DIR/png/${filename}.png"

        # Конвертация в JPEG (с потерями)
        convert -density 150 "$svg_file" -background white -flatten \
            "$OUTPUT_DIR/jpg/${filename}.jpg"

        # Конвертация в TIFF (для печати)
        convert -density 300 "$svg_file" "$OUTPUT_DIR/tiff/${filename}.tiff"

        echo "Обработан: $filename.svg"
    fi
done

echo "Пакетная конвертация завершена!"
```

3. Практическое задание "Конвейер обработки":

- Создать папку с 5 различными векторными файлами
- Написать скрипт для генерации 3 версий каждого файла
- Реализовать логирование процесса конвертации

Часть 4: Специализированные инструменты (≈2 часа)

Задача: Изучить продвинутые инструменты конвертации.

1. Использование XnConvert:

- Настройка входной и выходной папки
- Добавление действий:
  - Изменение размера
  - Коррекция цвета
  - Добавление водяных знаков
- Пакетная обработка

2. Онлайн-конвертеры:

```
python
# Пример использования API CloudConvert
import requests
import json

def convert_with_cloudconvert(input_file, output_format):
    api_key = "your_api_key"
    url = "https://api.cloudconvert.com/v2/jobs"

    payload = {
        "tasks": {
            "import-1": {
                "operation": "import/upload"
            },
            "convert-1": {
                "operation": "convert",
                "input": "import-1",
                "output_format": output_format,
                "engine": "inkscape"
            },
```

```
        "export-1": {
          "operation": "export/url",
          "input": "convert-1"
        }
      }
    }

    headers = {
      "Authorization": f"Bearer {api_key}",
      "Content-Type": "application/json"
    }

    response = requests.post(url, json=payload, headers=headers)
    return response.json()
```

3. Практическое задание "Сравнение инструментов":

- Выполнить конвертацию одного файла 3 способами
- Сравнить время выполнения и качество результата

Часть 5: Контроль качества и оптимизация (≈1 час)
Задача: Научиться оценивать и улучшать результаты конвертации.
1. Метрики качества:

```
python
from PIL import Image
import os

def analyze_conversion_quality(original_vector, converted_raster):
    """
    Анализ качества конвертации
    """
    with Image.open(converted_raster) as img:
        # Анализ размера
        width, height = img.size
        file_size = os.path.getsize(converted_raster)

        # Анализ цветов
        colors = img.getcolors(maxcolors=10000)

        print(f"Размер: {width}x{height} пикселей")
        print(f"Размер файла: {file_size / 1024:.2f} КБ")
        print(f"Количество цветов: {len(colors) if colors else 'Много'}")

        # Оценка качества
        if file_size > 1024 * 1024:  # > 1MB
            print("ВНИМАНИЕ: Большой размер файла")
        if width < 800:
            print("ВНИМАНИЕ: Низкое разрешение для веба")

# Использование
analyze_conversion_quality("diagram.svg", "diagram.png")
```

2. Оптимизация PNG файлов:

```
bash
# Использование pngquant для сжатия
pngquant --quality=80-95 --force diagram.png --output diagram-optimized.png

# Использование optipng для дополнительной оптимизации
optipng -o7 diagram-optimized.png
```

3. Создание превью разного качества:

```
bash
#!/bin/bash
# generate_previews.sh - Генерация превью разного качества

INPUT_FILE="$1"
FILENAME=$(basename "$INPUT_FILE" .svg)

# Высокое качество (для детального просмотра)
convert -density 300 "$INPUT_FILE" "previews/${FILENAME}_high.jpg"

# Среднее качество (для веб-сайта)
convert -density 150 "$INPUT_FILE" -resize 50% "previews/${FILENAME}_medium.jpg"

# Низкое качество (для превью)
convert -density 72 "$INPUT_FILE" -resize 25% "previews/${FILENAME}_low.jpg"

# PNG версия (без потерь)
convert -density 150 "$INPUT_FILE" "previews/${FILENAME}.png"
```

Итоговое задание для самостоятельного выполнения:

Сценарий: Вы получили задание подготовить графические материалы для корпоративного сайта и полиграфической продукции.

Задание:

1. **Подготовка исходных материалов:**
   - Создать/скачать 5 векторных файлов разных типов:
     - Логотип компании (.svg)
     - Организационная схема (.pdf)
     - Блок-схема процесса (.svg)
     - Техническая диаграмма (.ai)
     - Инфографика (.eps)
2. **Создание конвейера обработки:**
   - Написать скрипт для пакетной конвертации
   - Реализовать генерацию файлов для разных целей:
     - Веб-сайт (PNG, 72 DPI)
     - Презентации (JPEG, 150 DPI)
     - Полиграфия (TIFF, 300 DPI)
3. **Контроль качества:**
   - Создать таблицу сравнения результатов
   - Провести оптимизацию размеров файлов
   - Подготовить отчет о процессе

Требования к результату:

bash

```
final_project/
├── src/                    # Исходные векторные файлы
│   ├── logo.svg
│   ├── org-chart.pdf
│   ├── process-flow.svg
│   ├── technical-diagram.ai
│   └── infographic.eps
├── web/                    # Для веб-сайта
│   ├── logo.png
│   ├── org-chart.jpg
│   └── ...
├── print/                  # Для печати
│   ├── logo.tiff
│   ├── org-chart.tiff
│   └── ...
├── scripts/                # Скрипты обработки
│   ├── batch_convert.sh
│   ├── optimize_images.py
│   └── quality_check.py
└── report/                 # Отчеты
    ├── comparison_table.md
    ├── quality_analysis.txt
    └── process_log.txt
```

Критерии оценки:

- **Техническая корректность (30%):** Правильность параметров конвертации
- **Качество результатов (25%):** Сохранение читаемости и детализации
- **Автоматизация (20%):** Эффективность скриптов пакетной обработки
- **Оптимизация (15%):** Баланс между качеством и размером файлов
- **Документирование (10%):** Полнота отчетов и комментариев

Это занятие даст студентам практические навыки работы с профессиональными инструментами конвертации графики, что является важной компетенцией для веб-разработчиков, дизайнеров и технических писателей.

Практическое занятие 43 - 46. Отображение алгоритма или процесса с помощью блок-схемы, логическую структуру базы данных с помощью диаграммы "сущность - связь"

Отлично! Разработаем комплексный план практического занятия по созданию блок-схем алгоритмов и ER-диаграмм баз данных.

Практическое занятие 43-46: Создание блок-схем и ER-диаграмм

Цель занятия: Освоить принципы визуализации алгоритмов с помощью блок-схем и проектирования структуры баз данных с использованием диаграмм "сущность-связь".

Программное обеспечение:

- **Диаграммы:** Draw.io, Lucidchart, Visio
- **Бесплатные аналоги:** yEd Graph Editor, Dia
- **Текстовые генераторы:** Mermaid.js, PlantUML
- **Базы данных:** MySQL Workbench, dbdiagram.io

Необходимые знания: Базовое понимание алгоритмов и структуры данных, основы реляционных баз данных.

План занятия и Ход работы

Часть 1: Основы создания блок-схем (≈3 часа)

Задача: Освоить стандартные элементы блок-схем и принципы их построения.

1. Стандартные элементы блок-схем:

Элемент	Форма
Начало/Конец	Овал
Процесс	Прямоугольник
Решение	Ромб
Ввод/Вывод	Параллелограмм
Предопределенный процесс	Прямоугольник с двойными линиями
Соединитель	Круг

2. Практическое задание "Простой алгоритм":
3. Создание блок-схемы алгоритма авторизации:
4. Практическое задание "Сложный алгоритм":

- Создать блок-схему алгоритма обработки заказа в интернет-магазине:

text
```
[Получение заказа] → [Проверка наличия товара] → {Есть в наличии?}
  ↓ (Да)                    ↓ (Нет)
[Резервирование товара]      [Уведомление клиента]
  ↓                          ↓
[Формирование отгрузки] ← [Ожидание поставки] ← [Заказ у поставщика]
  ↓
[Отправка клиенту] → [Уведомление] → [Завершение заказа]
```

Часть 2: Профессиональное оформление блок-схем (≈2 часа)
Задача: Научиться создавать читаемые и профессионально оформленные схемы.
1. Принципы хорошей блок-схемы:

- **Логическая последовательность:** Один поток сверху вниз
- **Минимальная сложность:** Не более 10-15 элементов на схему
- **Четкие формулировки:** Лаконичные и понятные надписи
- **Визуальная иерархия:** Важные элементы выделены

2. Практическое задание в Draw.io:

- Создать блок-схему алгоритма сортировки массива
- Использовать различные стили для разных типов блоков
- Добавить поясняющие комментарии

3. Пример алгоритма поиска в массиве:

Часть 3: Основы ER-диаграмм (≈3 часа)
Задача: Понять концепции сущностей, атрибутов и связей в базах данных.
1. Основные компоненты ER-диаграмм:

Компонент	Обозначение	Описан
Сущность	Прямоугольник	Об
Атрибут	Овал	Св
Связь	Ромб	От
Ключ	Подчеркивание	Ун

2. Типы связей между сущностями:

- **Один-к-одному (1:1)** - |--|
- **Один-ко-многим (1:N)** - |--o{
- **Многие-ко-многим (N:M)** - }--{

3. Практическое задание "Простая база данных":

Часть 4: Создание сложных ER-диаграмм (≈3 часа)

Задача: Освоить проектирование нормализованных структур баз данных.

1. Диаграмма для системы управления библиотекой:

2. Практическое задание "База данных университета":

- Спроектировать ER-диаграмму для системы учета студентов:

text

```
[СТУДЕНТ] -- обучается на -- [ФАКУЛЬТЕТ]
[СТУДЕНТ] -- записан на -- [КУРС]
[ПРЕПОДАВАТЕЛЬ] -- преподает -- [КУРС]
[КУРС] -- включает -- [ЗАНЯТИЕ]
[СТУДЕНТ] -- имеет -- [ОЦЕНКА] -- по -- [КУРС]
```

3. Нормализация базы данных:

- **1NF:** Атомарность значений
- **2NF:** Зависимость от полного ключа
- **3NF:** Транзитивная независимость

Часть 5: Интеграция блок-схем и ER-диаграмм (≈1 час)

Задача: Научиться связывать алгоритмы работы с структурой данных.

1. Практическое задание "Полный цикл":

- Создать ER-диаграмму для интернет-магазина
- Разработать блок-схему процесса оформления заказа
- Показать взаимосвязь между структурой данных и алгоритмами

2. Пример интеграции:

Итоговое задание для самостоятельного выполнения:

Сценарий: Вы - разработчик, которому нужно спроектировать систему для онлайн-банкинга.

Задание:

1. Блок-схемы процессов:

- **Алгоритм регистрации пользователя**
- **Процесс денежного перевода**
- **Схема авторизации и аутентификации**

2. ER-диаграмма базы данных:

3. Требования к проекту:

1. **Блок-схемы:**
   - Минимум 3 различных алгоритма
   - Использование всех основных элементов блок-схем
   - Профессиональное оформление
2. **ER-диаграмма:**
   - Минимум 5 сущностей
   - Различные типы связей (1:1, 1:N, N:M)
   - Первичные и внешние ключи
   - Основные атрибуты для каждой сущности

4. Структура проекта:

text

```
banking-system-design/
├── flowcharts/
│   ├── user-registration.png
│   ├── money-transfer.png
│   └── authentication.png
├── database/
│   ├── er-diagram.png
│   ├── database-schema.sql
│   └── normalization-explanation.md
├── integration/
│   └── process-data-mapping.md
└── README.md
```

Критерии оценки:

Компонент	Критерии
Блок-схемы	Логическая корректность, читаемость, полнота
ER-диаграмма	Нормализация, типы связей, атрибуты
Интеграция	Связь алгоритмов со структурой данных
Оформление	Профессиональный вид, документация

Это занятие даст студентам фундаментальные навыки проектирования алгоритмов и структур данных, которые являются основой разработки программного обеспечения.

Практическое занятие 47 - 50. Отображение структуры ком-пьютерной сети или системы с помощью графической схемы

Отлично! Разработаем комплексный план практического занятия по созданию графических схем компьютерных сетей и систем.

Практическое занятие 47-50: Отображение структуры компьютерной сети или системы с помощью графической схемы

Цель занятия: Освоить принципы проектирования и визуализации структур компьютерных сетей и системных архитектур с использованием стандартных нотаций и символов.

Программное обеспечение:

- **Сетевые диаграммы:** Draw.io, Lucidchart, Visio
- **Специализированные:** yEd, Cisco Packet Tracer
- **Архитектурные диаграммы:** ArchiMate, Structurizr
- **Облачные архитектуры:** Cloudcraft, Diagrams.net

Необходимые знания: Базовое понимание сетевых технологий, архитектуры компьютеров и принципов организации ИТ-инфраструктуры.

План занятия и Ход работы

Часть 1: Основы сетевых диаграмм (≈3 часа)

Задача: Освоить стандартные обозначения и принципы создания сетевых диаграмм.

1. Стандартные сетевые символы:

Устройство	Обозначение
Маршрутизатор	https://via.placeholder.com/40x20/4CAF50/FFFFFF?text=RT
Коммутатор	https://via.placeholder.com/40x20/2196F3/FFFFFF?text=SW
Сервер	https://via.placeholder.com/40x20/FF9800/FFFFFF?text=SVR
Рабочая станция	https://via.placeholder.com/40x20/9C27B0/FFFFFF?text=PC
Межсетевой экран	https://via.placeholder.com/40x20/F44336/FFFFFF?text=FW

2. Типы сетевых диаграмм:

- **Логическая схема:** Фокусируется на IP-адресации и VLAN
- **Физическая схема:** Отображает реальное расположение оборудования
- **Архитектурная схема:** Показывает взаимодействие систем

3. Практическое задание "Простая офисная сеть":

Часть 2: Создание комплексных сетевых схем (≈4 часа)

Задача: Научиться проектировать многоуровневые сетевые инфраструктуры.

1. Трехуровневая сетевая архитектура:

2. Практическое задание "Сеть предприятия":

- Спроектировать сеть для компании с 3 отделами:
  - Отдел разработки (50 пользователей)
  - Отдел маркетинга (30 пользователей)
  - Бухгалтерия (20 пользователей)
- Реализовать VLAN для изоляции трафика
- Настроить DMZ для веб-серверов

3. Схема с IP-адресацией:

Часть 3: Системные архитектурные диаграммы (≈3 часа)

Задача: Освоить создание диаграмм программных систем и их взаимодействия.

1. Диаграмма компонентов веб-приложения:

2. Практическое задание "Архитектура микросервисов":

- Спроектировать систему на основе микросервисов:
  - Сервис аутентификации
  - Сервис пользователей
  - Сервис заказов
  - Сервис уведомлений
  - API Gateway

Часть 4: Облачные архитектуры (≈3 часа)

Задача: Научиться проектировать схемы облачных инфраструктур.

1. AWS архитектура веб-приложения:

2. Практическое задание "Гибридная облачная инфраструктура":

- Спроектировать гибридную среду:
  - Локальный дата-центр
  - Публичное облако (AWS/Azure)
  - Соединение через VPN/ExpressRoute
  - Распределенная балансировка нагрузки

Часть 5: Создание документации и легенд (≈1 час)
Задача: Научиться создавать профессиональную документацию к схемам.
1. Легенда для сетевых диаграмм:
2. Практическое задание "Комплексная документация":

- Создать набор схем для ИТ-инфраструктуры компании
- Разработать единую легенду для всех диаграмм
- Подготовить пояснительную записку

Итоговое задание для самостоятельного выполнения:
Сценарий: Вы - сетевой архитектор в растущей IT-компании. Необходимо спроектировать полную инфраструктуру для нового офиса.
Задание:
1. Физическая сетевая схема:
2. Логическая схема сети:

- VLAN 10: Управление сетью (192.168.10.0/24)
- VLAN 20: Серверы (192.168.20.0/24)
- VLAN 30: Пользователи (192.168.30.0/24)
- VLAN 40: Гости (192.168.40.0/24)
- VLAN 50: VoIP (192.168.50.0/24)

3. Системная архитектура:

- Active Directory для аутентификации
- Файловый сервер с квотами
- Сервер резервного копирования
- Система мониторинга
- Сервер печати

4. Требования к проекту:

1. **Сетевая схема:**
   - Отображение всех критических компонентов
   - Резервирование ключевых элементов
   - Масштабируемость на 100+ пользователей
2. **Документация:**
   - Легенда с обозначениями
   - IP-план сети
   - Пояснительная записка
3. **Безопасность:**
   - Сегментация сети
   - Политики доступа
   - Система мониторинга

5. Структура проекта:
text

```
network-design/
├── diagrams/
│   ├── physical-network.png
│   ├── logical-network.png
│   ├── ip-scheme.png
│   └── system-architecture.png
├── documentation/
│   ├── legend.md
│   ├── ip-plan.csv
│   ├── equipment-list.md
│   └── implementation-plan.md
└── specifications/
    ├── security-policy.md
    ├── backup-strategy.md
    └── monitoring-setup.md
```

Критерии оценки:

Компонент	Критерии
Сетевая схема	Полнота, корректность, масштабируемость
Логическая структура	IP-планирование, VLAN, маршрутизация

Компонент	Критерии
Системная архитектура	Взаимодействие компонентов, отказоустойчивость
Документация	Профессиональное оформление, полнота
Безопасность	Сегментация, политики доступа

Это занятие даст студентам практические навыки проектирования комплексных ИТ-инфраструктур, что является ключевой компетенцией для сетевых администраторов и системных архитекторов.
This response is AI-generated, for reference only.
Практическое занятие 51 - 54. Получение иллюстративные снимки экрана.
Отлично! Разработаем комплексный план практического занятия по созданию профессиональных иллюстративных снимков экрана.

Практическое занятие 51-54: Получение иллюстративных снимков экрана
Цель занятия: Освоить профессиональные методы создания, редактирования и аннотирования снимков экрана для технической документации, руководств пользователя и презентаций.
Программное обеспечение:

- **Захват экрана:** ShareX (Windows), Shutter (Linux), CleanShot X (Mac)
- **Редактирование:** GIMP, Paint.NET, Adobe Photoshop
- **Аннотации:** Snagit, Lightshot, Nimbus Screenshot
- **Организация:** Greenshot, PicPick
- **Встроенные средства:** Snipping Tool, Windows + Shift + S

Необходимые знания: Базовые навыки работы с операционной системой, понимание принципов визуальной коммуникации.

План занятия и Ход работы
Часть 1: Основы создания снимков экрана (≈3 часа)
Задача: Освоить различные методы захвата экрана и выбор подходящего инструмента.
1. Методы захвата экрана:

Метод	Сочетание клавиш
Весь экран	Print Screen
Активное окно	Alt + Print Screen
Выделенная область	Win + Shift + S (Windows)
Окно с меню	Специальные инструменты
Прокручиваемая область	Инструменты браузера

2. Настройка ShareX для профессиональной работы:
yaml
```
# Конфигурация горячих клавиш
hotkeys:
  rectangular_region: Ctrl+Shift+1
  active_window: Ctrl+Shift+2
  full_screen: Ctrl+Shift+3
  scrolling_capture: Ctrl+Shift+4

# Настройки после захвата
after_capture_actions:
  - add_image_effects
  - copy_image_to_clipboard
  - save_image_to_file

# Настройки изображения
image_settings:
  format: png
  jpeg_quality: 95
  reduce_colors: false
```

3. Практическое задание "Базовые снимки":

- Сделать снимок всего рабочего стола
- Захватить активное окно браузера
- Выделить и захватить область с панелью инструментов
- Создать снимок с задержкой 5 секунд

Часть 2: Специализированные техники захвата (≈4 часа)
Задача: Освоить продвинутые методы захвата для различных сценариев.
1. Захват выпадающих меню:
python
 # Алгоритм захвата меню
1. Активировать программу захвата
2. Установить задержку 3-5 секунд
3. Открыть нужное меню
4. Дождаться автоматического захвата
5. Сохранить результат
2. Захват длинных веб-страниц:

- **Браузерные расширения:** GoFullPage, Nimbus Screenshot
- **Встроенные средства разработчика:** Ctrl+Shift+P → "screenshot"
- **Специализированные инструменты:** PicPick, Lightshot

3. Практическое задание "Сложные сценарии":
markdown
Задание 1: Захват контекстного меню
- Открыть проводник
- Захватить контекстное меню правой кнопки мыши
- Сохранить с прозрачным фоном

Задание 2: Захват веб-страницы
- Открыть длинную статью в браузере
- Сделать полноэкранный скриншот всей страницы
- Сохранить в PDF и PNG

Задание 3: Захват анимации
- Записать GIF работы интерфейса
- Показать процесс заполнения формы
- Экспортировать в оптимизированный GIF
4. Создание скринкастов:
bash
 # Использование OBS Studio для записи экрана
1. Настроить область захвата
2. Выбрать источник звука (микрофон/система)
3. Установить качество записи
4. Записать процесс работы
5. Экспортировать в MP4/GIF

Часть 3: Редактирование и аннотирование (≈4 часа)
Задача: Научиться профессионально обрабатывать и аннотировать скриншоты.
1. Основные типы аннотаций:

Аннотация	Назначение
Нумерация	Последовательность шагов
Стрелки	Указание на элементы
Выделение	Акцентирование внимания
Размытие	Сокрытие конфиденциальной информации
Текст	Пояснения

2. Практическое задание "Аннотирование интерфейса":
3. Создание шаблонов аннотаций в GIMP:
python
 # Псевдокод для создания шаблонов аннотаций
def create_annotation_template():
    # Создать слои для различных типов аннотаций
    layers = {
        'red_arrow': create_arrow_layer(color='#FF0000'),

```
        'blue_highlight': create_rectangle_layer(color='#0000FF', opacity=0.3),
        'number_circle': create_circle_layer(color='#00FF00'),
        'text_callout': create_text_layer(style='bold', size=14)
    }
    return layers
```

4. Пакетная обработка скриншотов:

```
bash
#!/bin/bash
# batch_annotate.sh - Пакетное добавление водяных знаков

INPUT_DIR="raw_screenshots"
OUTPUT_DIR="annotated_screenshots"
WATERMARK="company_logo.png"

for file in "$INPUT_DIR"/*.png; do
    if [ -f "$file" ]; then
        filename=$(basename "$file")

        # Добавление водяного знака
        convert "$file" "$WATERMARK" -gravity southeast -geometry +10+10 \
            -composite "$OUTPUT_DIR/watermarked_$filename"

        # Добавление рамки
        convert "$OUTPUT_DIR/watermarked_$filename" -bordercolor white \
            -border 10x10 "$OUTPUT_DIR/final_$filename"

        echo "Обработан: $filename"
    fi
done
```

Часть 4: Организация и именование файлов (≈2 часа)

Задача: Создать эффективную систему организации скриншотов.

1. Структура папок для проекта:

```
text
screenshot-project/
├── 01-raw-captures/          # Исходные скриншоты
│   ├── login-process/
│   ├── user-dashboard/
│   └── admin-panel/
├── 02-annotated/             # Аннотированные версии
│   ├── step-by-step/
│   ├── highlighted/
│   └── numbered/
├── 03-optimized/             # Оптимизированные для публикации
│   ├── web/                  # 72 DPI
│   ├── print/                # 300 DPI
│   └── mobile/               # 150 DPI
├── 04-screencasts/           # Видеозаписи экрана
│   ├── gif/
│   ├── mp4/
│   └── webm/
└── metadata/                 # Метаданные
    ├── descriptions.csv
    └── version-history.md
```

2. Система именования файлов:

```
python
# Генератор имен файлов
def generate_screenshot_name(app_name, feature, step, resolution, language):
    template = f"{app_name}_{feature}_step{step:02d}_{resolution}_{language}"
    return template.lower().replace(' ', '_')

# Примеры имен:
# - word_document_creation_step01_1920x1080_ru.png
# - excel_data_analysis_step05_800x600_en.jpg
# - browser_navigation_step03_fullpage_ru.png
```

3. Практическое задание "Организация проекта":

- Создать структуру папок для документации программного обеспечения
- Разработать систему именования для 20+ скриншотов
- Создать таблицу метаданных с описанием каждого скриншота

Часть 5: Создание иллюстративных последовательностей (≈3 часа)

Задача: Научиться создавать последовательности скриншотов для пошаговых руководств.

1. Планирование последовательности:

```
markdown
## Процесс установки программного обеспечения
```

Шаг 1: Запуск установщика
- Скриншот: installer_welcome.png
- Описание: Окно приветствия установщика

Шаг 2: Выбор директории
- Скриншот: install_directory.png
- Описание: Выбор папки для установки

Шаг 3: Настройка компонентов
- Скриншот: components_selection.png
- Описание: Выбор устанавливаемых компонентов

Шаг 4: Завершение установки
- Скриншот: installation_complete.png
- Описание: Окно успешной установки

2. Создание последовательности в Draw.io:

3. Практическое задание "Пошаговое руководство":

- Создать 10 скриншотов процесса регистрации на сайте
- Добавить последовательную нумерацию
- Разместить в коллаж для обзора
- Экспортировать в PDF-документ

Итоговое задание для самостоятельного выполнения:

Сценарий: Вы - технический писатель в software company. Вам нужно создать полный комплект иллюстраций для руководства пользователя нового приложения.

Задание:

1. Захват основных функций приложения:

- **Авторизация:** 5 скриншотов процесса входа
- **Главное меню:** 3 скриншота различных состояний
- **Работа с данными:** 8 скриншотов CRUD операций
- **Настройки:** 4 скриншота конфигурации

2. Создание аннотированных версий:

python
```
# Пример структуры аннотаций
annotations = {
    'login_sequence': [
        {'step': 1, 'annotation': 'Введите email', 'element': 'email_field'},
        {'step': 2, 'annotation': 'Введите пароль', 'element': 'password_field'},
        {'step': 3, 'annotation': 'Нажмите кнопку Входа', 'element': 'login_button'}
    ],
    'dashboard_elements': [
        {'element': 'sidebar', 'annotation': 'Основная навигация'},
        {'element': 'search_bar', 'annotation': 'Поиск по данным'},
        {'element': 'user_profile', 'annotation': 'Управление профилем'}
    ]
}
```

3. Организация файлов:

text
```
user-manual-screenshots/
├── 01-raw/
│   ├── authentication/
│   │   ├── login_window.png
│   │   ├── password_recovery.png
│   │   └── two_factor.png
│   ├── main-interface/
│   │   ├── dashboard.png
│   │   ├── navigation.png
│   │   └── settings.png
│   └── features/
│       ├── data_import.png
│       ├── report_generation.png
│       └── export_options.png
├── 02-annotated/
│   ├── numbered-steps/
│   │   ├── login_process/
│   │   └── data_export/
│   ├── highlighted/
│   │   ├── important_buttons/
│   │   └── menu_items/
│   └── collages/
│       ├── interface_overview.png
│       └── workflow_summary.png
├── 03-optimized/
│   ├── web/                # 72 DPI, JPEG
│   ├── print/              # 300 DPI, PNG
│   └── mobile/             # 150 DPI, WebP
```

```
└── documentation/
    ├── screenshot_index.csv
    ├── annotation_guide.md
    └── style_guide.md
```

4. Требования к проекту:

1. **Полнота охвата:** Все основные функции приложения
2. **Качество изображений:** Четкость, правильное разрешение
3. **Профессиональные аннотации:** Ясность, единый стиль
4. **Организация:** Логичная структура папок и именование
5. **Документация:** Описания и метаданные для каждого скриншота

Критерии оценки:

Компонент	Критерии
Техническое качество	Четкость, разрешение, формат
Аннотации и разметка	Ясность, единый стиль, точность
Организация	Структура папок, именование файлов
Полнота покрытия	Все ключевые функции приложения
Документация	Метаданные, описания, руководства

Это занятие даст студентам профессиональные навыки создания качественных иллюстративных материалов, которые необходимы для технической документации, руководств пользователя и обучающих материалов.
Практическое занятие 55 - 58. Оптимизация снимка экрана для размещения на странице заданного формата
Отлично! Разработаем комплексный план практического занятия по оптимизации снимков экрана для различных форматов публикации.

Практическое занятие 55-58: Оптимизация снимка экрана для размещения на странице заданного формата
Цель занятия: Освоить профессиональные методы оптимизации скриншотов для различных форматов публикации (веб, печать, мобильные устройства) с сохранением качества и читаемости.
Программное обеспечение:

- **Редакторы изображений:** GIMP, Adobe Photoshop, Paint.NET
- **Инструменты оптимизации:** ImageOptim, TinyPNG, Squoosh
- **Пакетная обработка:** XnConvert, IrfanView
- **Командные утилиты:** ImageMagick, FFmpeg
- **Онлайн-инструменты:** Compressor.io, ShortPixel

Необходимые знания: Базовые навыки работы со скриншотами, понимание форматов изображений.

План занятия и Ход работы
Часть 1: Основы оптимизации изображений (≈3 часа)
Задача: Понять принципы оптимизации и критерии выбора форматов.
1. Параметры оптимизации:

Параметр	Влияние
Разрешение (DPI)	Четкость при печати
Размер в пикселях	Отображение на экране
Формат файла	Качество и размер
Степень сжатия	Баланс качество/размер
Цветовая палитра	Размер файла

2. Матрица выбора формата:

Сценарий использования	Рекомендуемый формат	Настройки	Причина
Скриншот с текстом	PNG	Без потерь	Сохранение четкости текста
Фотография в интерфейсе	JPEG	Качество 75%	Эффективное сжатие
Веб-сайт	WebP	Качество 80%	Современный формат
Печатная документация	TIFF	300 DPI	Высокое качество печати
Мобильное приложение	PNG/WebP	Разные density	Адаптивность

3. Практическое задание "Анализ исходников":

```
python
from PIL import Image
import os

def analyze_screenshot(file_path):
    """Анализ параметров скриншота"""
    with Image.open(file_path) as img:
        info = {
            'format': img.format,
            'size_pixels': img.size,
            'mode': img.mode,
            'file_size_mb': os.path.getsize(file_path) / 1024 / 1024,
            'dpi': img.info.get('dpi', (72, 72))
        }
        return info

# Пример использования
analysis = analyze_screenshot('raw_screenshot.png')
print(f"Размер: {analysis['size_pixels']}")
print(f"Размер файла: {analysis['file_size_mb']:.2f} МБ")
print(f"DPI: {analysis['dpi']}")
```

Часть 2: Оптимизация для веб-форматов (≈4 часа)
Задача: Научиться готовить скриншоты для веб-сайтов и онлайн-документации.
1. Оптимизация PNG для веба:

```
bash
#!/bin/bash
# optimize_png.sh - Оптимизация PNG скриншотов

INPUT_DIR="raw_screenshots"
OUTPUT_DIR="web_optimized"
MAX_WIDTH=1200
QUALITY=85

for file in "$INPUT_DIR"/*.png; do
    if [ -f "$file" ]; then
        filename=$(basename "$file" .png)

        # Изменение размера и оптимизация
        convert "$file" -resize "${MAX_WIDTH}x>" \
                -strip \
                -quality $QUALITY \
                "$OUTPUT_DIR/${filename}_web.png"

        # Дополнительная оптимизация с pngquant
        pngquant --quality=80-95 --force \
            "$OUTPUT_DIR/${filename}_web.png" \
            --output "$OUTPUT_DIR/${filename}_optimized.png"

        echo "Оптимизирован: $filename"
    fi
done
```

2. Создание responsive изображений:

```
html
<!-- HTML для responsive скриншотов -->
```

```
<picture>
  <source media="(max-width: 768px)"
          srcset="screenshot-mobile.webp 1x, screenshot-mobile@2x.webp 2x">
  <source media="(max-width: 1200px)"
          srcset="screenshot-tablet.webp 1x, screenshot-tablet@2x.webp 2x">
  <img src="screenshot-desktop.webp"
       alt="Описание скриншота"
       loading="lazy">
</picture>
```

3. Практическое задание "Веб-галерея":

- Подготовить 10 скриншотов для галереи документации
- Создать 3 версии каждого (mobile, tablet, desktop)
- Оптимизировать размеры и форматы
- Сгенерировать HTML-галерею

Часть 3: Подготовка для печатных форматов (≈4 часа)
Задача: Освоить подготовку скриншотов для полиграфии.
1. Расчет размеров для печати:

```
python
def calculate_print_size(screenshot_path, target_dpi=300):
    """Расчет физического размера для печати"""
    from PIL import Image

    with Image.open(screenshot_path) as img:
        width_px, height_px = img.size
        current_dpi = img.info.get('dpi', (72, 72))[0]

        # Конвертация в дюймы и сантиметры
        width_inch = width_px / target_dpi
        height_inch = height_px / target_dpi
        width_cm = width_inch * 2.54
        height_cm = height_inch * 2.54

        return {
            'pixels': (width_px, height_px),
            'inches': (round(width_inch, 2), round(height_inch, 2)),
            'centimeters': (round(width_cm, 2), round(height_cm, 2)),
            'max_print_size': f"{width_cm:.1f}×{height_cm:.1f} см"
        }

# Пример расчета
size_info = calculate_print_size('screenshot.png')
print(f"Максимальный размер печати: {size_info['max_print_size']}")
```

2. Скрипт подготовки для печати:

```
bash
#!/bin/bash
# prepare_for_print.sh - Подготовка скриншотов для печати

INPUT_DIR="raw_screenshots"
OUTPUT_DIR="print_ready"
TARGET_DPI=300
COLOR_PROFILE="CMYK"

for file in "$INPUT_DIR"/*.png; do
    if [ -f "$file" ]; then
        filename=$(basename "$file" .png)

        # Конвертация в TIFF с высоким DPI
        convert "$file" -density $TARGET_DPI \
                -colorspace $COLOR_PROFILE \
                -units PixelsPerInch \
                "$OUTPUT_DIR/${filename}_print.tiff"

        # Создание PDF версии
        convert "$file" -density $TARGET_DPI \
                "$OUTPUT_DIR/${filename}_print.pdf"

        echo "Подготовлен для печати: $filename"
    fi
done
```

3. Практическое задание "Печатное руководство":

- Подготовить скриншоты для руководства формата A4
- Рассчитать оптимальные размеры изображений
- Конвертировать в CMYK цветовое пространство
- Создать PDF с embedded изображениями

Часть 4: Оптимизация для мобильных устройств (≈3 часа)
Задача: Научиться адаптировать скриншоты для мобильных платформ.
1. Плотности пикселей для мобильных:
python
```
# Конфигурация плотностей для Android/iOS
DENSITY_CONFIG = {
    'android': {
        'ldpi': 0.75,
        'mdpi': 1.0,
        'hdpi': 1.5,
        'xhdpi': 2.0,
        'xxhdpi': 3.0,
        'xxxhdpi': 4.0
    },
    'ios': {
        '@1x': 1.0,
        '@2x': 2.0,
        '@3x': 3.0
    }
}

def generate_mobile_assets(base_image, platform, base_size):
    """Генерация assets для мобильных платформ"""
    assets = {}

    for density_name, multiplier in DENSITY_CONFIG[platform].items():
        new_size = (int(base_size[0] * multiplier),
                    int(base_size[1] * multiplier))

        # Масштабирование изображения
        scaled_image = base_image.resize(new_size, Image.Resampling.LANCZOS)

        filename = f"asset_{density_name}.png"
        assets[density_name] = {
            'image': scaled_image,
            'filename': filename,
            'size': new_size
        }

    return assets
```
2. Скрипт для создания мобильных assets:
bash
```
#!/bin/bash
# create_mobile_assets.sh - Создание assets для мобильных приложений

INPUT_FILE="app_screenshot.png"
OUTPUT_DIR="mobile_assets"

# Базовый размер для @1x
BASE_WIDTH=375  # iPhone стандарт

# Создание версий для разных плотностей
convert "$INPUT_FILE" -resize "${BASE_WIDTH}x" "$OUTPUT_DIR/ios_1x.png"
convert "$INPUT_FILE" -resize "$((BASE_WIDTH * 2))x" "$OUTPUT_DIR/ios_2x.png"
convert "$INPUT_FILE" -resize "$((BASE_WIDTH * 3))x" "$OUTPUT_DIR/ios_3x.png"

# Android версии
convert "$INPUT_FILE" -resize "$((BASE_WIDTH * 1))x" "$OUTPUT_DIR/android_mdpi.png"
convert "$INPUT_FILE" -resize "$((BASE_WIDTH * 1.5))x" "$OUTPUT_DIR/android_hdpi.png"
convert "$INPUT_FILE" -resize "$((BASE_WIDTH * 2))x" "$OUTPUT_DIR/android_xhdpi.png"
```
3. Практическое задание "Мобильный store":
- Подготовить скриншоты для App Store и Google Play
- Создать наборы для различных размеров экранов
- Оптимизировать для быстрой загрузки
- Проверить читаемость на реальных устройствах

Часть 5: Пакетная обработка и автоматизация (≈2 часа)
Задача: Создать автоматизированный пайплайн обработки скриншотов.
1. Конфигурационный файл для пакетной обработки:
yaml
```
# optimization_config.yaml
project: "user_manual"
formats:
  web:
    output_dir: "web"
    formats: ["webp", "jpg"]
```

```
    max_width: 1200
    quality: 80
    dpi: 72

  print:
    output_dir: "print"
    formats: ["tiff", "pdf"]
    dpi: 300
    color_space: "CMYK"

  mobile:
    output_dir: "mobile"
    densities: ["1x", "2x", "3x"]
    base_width: 375

optimization:
  png_compression_level: 9
  jpeg_quality: 85
  strip_metadata: true
  progressive_jpeg: true
```

2. Python скрипт для пакетной обработки:

```
python
import yaml
from PIL import Image
import os
from pathlib import Path

class ScreenshotOptimizer:
    def __init__(self, config_path):
        with open(config_path, 'r', encoding='utf-8') as f:
            self.config = yaml.safe_load(f)

    def optimize_screenshot(self, input_path, output_format):
        """Оптимизация одного скриншота"""
        with Image.open(input_path) as img:
            # Применение настроек из конфигурации
            if output_format == 'web':
                return self._prepare_web_version(img, input_path)
            elif output_format == 'print':
                return self._prepare_print_version(img, input_path)
            elif output_format == 'mobile':
                return self._prepare_mobile_version(img, input_path)

    def batch_optimize(self, input_dir):
        """Пакетная оптимизация всех скриншотов в директории"""
        input_path = Path(input_dir)

        for screenshot_file in input_path.glob('*.png'):
            for format_type in self.config['formats']:
                self.optimize_screenshot(screenshot_file, format_type)

        print("Пакетная оптимизация завершена!")

# Использование
optimizer = ScreenshotOptimizer('optimization_config.yaml')
optimizer.batch_optimize('raw_screenshots')
```

Итоговое задание для самостоятельного выполнения:

Сценарий: Вы - технический специалист, которому нужно подготовить комплект скриншотов для многоформатного проекта.

Задание:

1. Подготовка исходных материалов:

- Собрать 15 скриншотов веб-приложения
- Включить интерфейсы, диаграммы, таблицы данных
- Сохранить в максимальном качестве

2. Создание оптимизированных версий:

```
bash
project_optimization/
├── 01-source/              # Исходные скриншоты
│   ├── login_interface.png
│   ├── user_dashboard.png
│   ├── admin_panel.png
│   └── ...
├── 02-web/                 # Веб-версии
│   ├── webp/               # Modern format
│   ├── jpg/                # Fallback
```

```
│   └── responsive/          # Разные размеры
├── 03-print/                # Печатные версии
│   ├── tiff/                # Высокое качество
│   ├── pdf/                 # Для документации
│   └── a4_layout/           # Готовые к печати
├── 04-mobile/               # Мобильные версии
│   ├── ios/
│   ├── android/
│   └── responsive/
└── 05-reports/              # Отчеты и аналитика
    ├── size_comparison.csv
    ├── quality_analysis.md
    └── optimization_log.txt
```

3. Требования к оптимизации:

Для веб-форматов:

- Максимальная ширина: 1200px
- Форматы: WebP (основной), JPEG (fallback)
- Размер файла: < 200KB каждый
- DPI: 72

Для печатных форматов:

- Разрешение: 300 DPI
- Цветовая модель: CMYK
- Форматы: TIFF, PDF
- Поддержка стандартов полиграфии

Для мобильных устройств:

- Плотности: 1x, 2x, 3x
- Форматы: PNG
- Оптимизация под Retina дисплеи

4. Автоматизация процесса:

```
python
# comprehensive_optimizer.py
def generate_optimization_report(input_dir, output_dir):
    """Генерация отчета об оптимизации"""
    report = {
        'total_files': 0,
        'total_size_before': 0,
        'total_size_after': 0,
        'formats_created': [],
        'quality_metrics': {}
    }

    # Реализация сбора статистики
    # ...

    return report

# Создание сравнительной таблицы
def create_comparison_table(original_files, optimized_files):
    """Создание таблицы сравнения до/после"""
    comparison_data = []

    for orig, opt in zip(original_files, optimized_files):
        orig_size = os.path.getsize(orig) / 1024
        opt_size = os.path.getsize(opt) / 1024
        savings = ((orig_size - opt_size) / orig_size) * 100

        comparison_data.append({
            'filename': os.path.basename(orig),
            'original_size_kb': round(orig_size, 2),
            'optimized_size_kb': round(opt_size, 2),
            'savings_percent': round(savings, 1),
            'format': os.path.splitext(opt)[1]
        })

    return comparison_data
```

5. Критерии оценки:

Компонент	Критерии
Качество оптимизации	Сохранение читаемости, размер файлов
Многоформатность	Поддержка всех целевых форматов

Компонент	Критерии
Автоматизация	Скрипты пакетной обработки
Организация	Структура файлов, именование
Документация	Отчеты, метрики, анализ

Метрики успеха:
- Снижение общего размера файлов на 60-80%
- Сохранение читаемости текста и интерфейсов
- Корректное отображение на всех целевых платформах
- Профессиональная организация файлов

Это занятие даст студентам профессиональные навыки подготовки графических материалов для публикации в различных форматах, что является критически важным для веб-разработчиков, технических писателей и дизайнеров.
Практическое занятие 59 - 62. Оптимизация цветности снимка экрана для воспроизведения заданным способом. Нанесение на снимок экрана поясняющие надписи и графические элементы
Отлично! Разработаем комплексный план практического занятия по оптимизации цветности и аннотированию скриншотов.

Практическое занятие 59-62: Оптимизация цветности и аннотирование снимков экрана
Цель занятия: Освоить профессиональные методы оптимизации цветовых профилей для различных устройств вывода и научиться создавать информативные аннотации с поясняющими элементами.
Программное обеспечение:
- **Редакторы:** GIMP, Adobe Photoshop, Krita
- **Инструменты цвета:** ColorSync, DisplayCAL
- **Аннотации:** Snagit, Monosnap, Lightshot
- **Векторная графика:** Inkscape, Figma
- **Аналитика:** ImageMagick, ExifTool

Необходимые знания: Базовые навыки работы со скриншотами, понимание цветовых моделей.

План занятия и Ход работы
Часть 1: Основы управления цветом (≈3 часа)
Задача: Понять принципы цветовых пространств и их оптимизации для разных устройств.
1. Цветовые модели и их применение:

Цветовая модель	Применение
sRGB	Веб-браузеры, офисные приложения
Adobe RGB	Профессиональная фотография
CMYK	Полиграфия, печать
P3	Современные дисплеи Apple
Grayscale	ЧБ печать, документы

2. Практическое задание "Анализ цветовых профилей":
python
```
from PIL import Image, ImageCms
import os

def analyze_color_profile(image_path):
    """Анализ цветового профиля изображения"""
    with Image.open(image_path) as img:
        info = {
            'format': img.format,
            'mode': img.mode,
            'icc_profile': img.info.get('icc_profile'),
            'color_space': 'Unknown'
        }

        # Определение цветового пространства
```

```
    if info['icc_profile']:
        try:
            profile = ImageCms.ImageCmsProfile(io.BytesIO(info['icc_profile']))
            info['color_space'] = profile.profile.profile_description
        except:
            info['color_space'] = 'Embedded (unknown)'
    elif img.mode == 'RGB':
        info['color_space'] = 'sRGB (assumed)'
    elif img.mode == 'CMYK':
        info['color_space'] = 'CMYK'

    return info

# Пример использования
analysis = analyze_color_profile('screenshot.png')
print(f"Цветовое пространство: {analysis['color_space']}")
```

3. Конвертация между цветовыми пространствами:

```
bash
#!/bin/bash
# convert_color_spaces.sh - Конвертация цветовых пространств

INPUT_FILE="screenshot.png"

# Конвертация в sRGB для веба
convert "$INPUT_FILE" -profile "sRGB.icc" "screenshot_srgb.jpg"

# Конвертация в CMYK для печати
convert "$INPUT_FILE" -profile "USWebCoatedSWOP.icc" "screenshot_cmyk.tiff"

# Конвертация в Grayscale
convert "$INPUT_FILE" -colorspace Gray "screenshot_grayscale.png"

# Создание версии с уменьшенной палитрой
convert "$INPUT_FILE" -colors 64 "screenshot_optimized.png"
```

Часть 2: Оптимизация цветности для различных носителей (≈4 часа)

Задача: Научиться готовить скриншоты для конкретных устройств вывода.

1. Настройка цветности для веб-публикации:

```
python
from PIL import Image, ImageCms

def optimize_for_web(input_path, output_path, quality=85):
    """Оптимизация скриншота для веб-публикации"""

    with Image.open(input_path) as img:
        # Конвертация в sRGB если нужно
        if img.mode != 'RGB':
            img = img.convert('RGB')

        # Применение sRGB профиля
        srgb_profile = ImageCms.createProfile('sRGB')

        if 'icc_profile' in img.info:
            # Конвертация в sRGB
            img = ImageCms.profileToProfile(img,
                                ImageCms.ImageCmsProfile(io.BytesIO(img.info['icc_profile'])),
                                srgb_profile)
        else:
            # Предполагаем sRGB и встраиваем профиль
            img.info['icc_profile'] = srgb_profile.tobytes()

        # Оптимизация и сохранение
        img.save(output_path, 'JPEG',
                 quality=quality,
                 optimize=True,
                 progressive=True,
                 icc_profile=img.info['icc_profile'])

# Использование
optimize_for_web('raw_screenshot.png', 'web_optimized.jpg', quality=80)
```

2. Практическое задание "Адаптация для различных носителей":

```
markdown
### Задание 1: Социальные сети
- Платформа: Instagram, Facebook
- Цветовое пространство: sRGB
- Размер: 1080x1080px
```

- Особенности: Яркие, контрастные цвета

Задание 2: Профессиональная печать
- Носитель: Глянцевая бумага
- Цветовое пространство: CMYK
- Разрешение: 300 DPI
- Особенности: Учет растискивания точки

Задание 3: Электронные книги
- Устройство: E-ink дисплеи
- Цветовое пространство: Grayscale
- Контрастность: Повышенная
- Особенности: Оптимизация для ЧБ

3. Создание пресетов оптимизации:

python
```
class ColorOptimizationPresets:
    """Пресеты оптимизации цветности для различных целей"""

    @staticmethod
    def social_media_optimize(image):
        """Оптимизация для социальных сетей"""
        # Повышение насыщенности и контраста
        return image.convert('RGB').enhance(1.1).enhance(1.2)

    @staticmethod
    def print_optimize(image):
        """Оптимизация для печати"""
        # Конвертация в CMYK с правильным профилем
        return image.convert('CMYK')

    @staticmethod
    def ebook_optimize(image):
        """Оптимизация для электронных книг"""
        # Улучшение читаемости в градациях серого
        grayscale = image.convert('L')
        return grayscale.point(lambda x: 0 if x < 64 else 255)
```

Часть 3: Создание поясняющих надписей (≈4 часа)

Задача: Освоить методы добавления профессиональных текстовых аннотаций.

1. Типографика для аннотаций:

python
```
from PIL import Image, ImageDraw, ImageFont

class TextAnnotation:
    """Класс для создания текстовых аннотаций"""

    def __init__(self, font_path="arial.ttf", base_size=14):
        self.font_path = font_path
        self.base_size = base_size

    def add_step_number(self, image, position, number, bg_color='#FF4444'):
        """Добавление нумерации шагов"""
        draw = ImageDraw.Draw(image)

        # Создание круга с номером
        bbox = [position[0]-15, position[1]-15, position[0]+15, position[1]+15]
        draw.ellipse(bbox, fill=bg_color)

        # Добавление текста
        font = ImageFont.truetype(self.font_path, 12)
        text_bbox = draw.textbbox((0, 0), str(number), font=font)
        text_width = text_bbox[2] - text_bbox[0]
        text_height = text_bbox[3] - text_bbox[1]

        text_position = (position[0] - text_width//2, position[1] - text_height//2)
        draw.text(text_position, str(number), fill='white', font=font)

        return image

    def add_callout(self, image, text, position, max_width=200):
        """Добавление выноски с текстом"""
        draw = ImageDraw.Draw(image)
        font = ImageFont.truetype(self.font_path, self.base_size)

        # Разбивка текста на строки
        words = text.split()
```

```
        lines = []
        current_line = []

        for word in words:
            test_line = ' '.join(current_line + [word])
            bbox = draw.textbbox((0, 0), test_line, font=font)
            width = bbox[2] - bbox[0]

            if width <= max_width:
                current_line.append(word)
            else:
                lines.append(' '.join(current_line))
                current_line = [word]

        if current_line:
            lines.append(' '.join(current_line))

        # Рисование фона и текста
        line_height = 20
        total_height = len(lines) * line_height

        # Фон
        bg_bbox = [position[0], position[1],
                   position[0] + max_width + 10,
                   position[1] + total_height + 10]
        draw.rectangle(bg_bbox, fill='#FFFFE0', outline='#000000')

        # Текст
        for i, line in enumerate(lines):
            text_position = (position[0] + 5, position[1] + 5 + i * line_height)
            draw.text(text_position, line, fill='#000000', font=font)

        return image
```

2. Практическое задание "Аннотирование интерфейса":

```
python
# Пример аннотирования веб-интерфейса
def annotate_web_interface(screenshot_path):
    """Аннотирование элементов веб-интерфейса"""
    image = Image.open(screenshot_path)
    annotator = TextAnnotation()

    # Добавление номеров шагов
    steps = [
        (100, 150, 1, "Поле ввода логина"),
        (100, 200, 2, "Поле ввода парска"),
        (300, 250, 3, "Кнопка входа"),
        (400, 100, 4, "Ссылка восстановления")
    ]

    for x, y, num, description in steps:
        image = annotator.add_step_number(image, (x, y), num)
        image = annotator.add_callout(image, description, (x + 30, y - 10))

    return image
```

Часть 4: Добавление графических элементов (≈4 часа)

Задача: Научиться использовать графические элементы для пояснения.

1. Библиотека графических аннотаций:

```
python
class GraphicAnnotations:
    """Класс для графических аннотаций"""

    def __init__(self):
        self.colors = {
            'red': '#FF4444',
            'green': '#44FF44',
            'blue': '#4444FF',
            'yellow': '#FFFF44',
            'highlight': '#FFFF0055'
        }

    def add_arrow(self, image, start_pos, end_pos, color='red', width=3):
        """Добавление стрелки"""
        draw = ImageDraw.Draw(image)

        # Линия
```

```
        draw.line([start_pos, end_pos], fill=color, width=width)

        # Наконечник стрелки
        arrow_size = 10
        # Расчет точек наконечника...

        return image

    def add_highlight(self, image, bbox, color='highlight'):
        """Выделение области"""
        draw = ImageDraw.Draw(image, 'RGBA')

        # Полупрозрачное выделение
        draw.rectangle(bbox, fill=self.colors[color])

        # Контур
        draw.rectangle(bbox, outline=self.colors['red'], width=2)

        return image

    def add_blur_effect(self, image, bbox, radius=10):
        """Размытие конфиденциальной информации"""
        x1, y1, x2, y2 = bbox

        # Выделение области для размытия
        region = image.crop(bbox)

        # Применение размытия
        blurred_region = region.filter(ImageFilter.GaussianBlur(radius))

        # Вставка обратно
        image.paste(blurred_region, bbox)

        return image

    def add_zoom_effect(self, image, source_bbox, target_pos, zoom_level=2):
        """Эффект увеличения области"""
        # Выделение области для увеличения
        source_region = image.crop(source_bbox)

        # Увеличение
        new_size = (source_region.width * zoom_level,
                    source_region.height * zoom_level)
        zoomed_region = source_region.resize(new_size, Image.Resampling.LANCZOS)

        # Вставка увеличенной области
        image.paste(zoomed_region, target_pos)

        # Добавление соединительной линии
        self.add_arrow(image,
                       (source_bbox[2], source_bbox[3]),
                       (target_pos[0], target_pos[1]))

        return image
```

2. Практическое задание "Создание инструкции":

```
python
def create_step_by_step_guide(base_screenshot):
    """Создание пошаговой инструкции со скриншотами"""
    steps = []

    # Шаг 1: Выделение меню
    step1 = base_screenshot.copy()
    annotator = GraphicAnnotations()
    step1 = annotator.add_highlight(step1, (50, 100, 200, 150))
    step1 = annotator.add_arrow(step1, (300, 200), (150, 125))
    steps.append(('Шаг 1: Откройте меню', step1))

    # Шаг 2: Выбор пункта
    step2 = base_screenshot.copy()
    step2 = annotator.add_highlight(step2, (150, 200, 300, 250))
    step2 = annotator.add_zoom_effect(step2, (150, 200, 300, 250), (400, 100))
    steps.append(('Шаг 2: Выберите пункт', step2))

    return steps
```

Часть 5: Комплексная обработка и экспорт (≈3 часа)

Задача: Создать автоматизированный пайплайн обработки скриншотов.

1. Конфигурация проекта:

yaml

```yaml
# annotation_config.yaml
project: "user_manual_annotations"

color_settings:
  web:
    color_space: "sRGB"
    optimization: "high_contrast"
    output_format: "jpg"

  print:
    color_space: "CMYK"
    optimization: "print_ready"
    output_format: "tiff"

annotations:
  step_numbers:
    enabled: true
    color: "#FF4444"
    font_size: 14

  callouts:
    enabled: true
    background: "#FFFFE0"
    max_width: 250

  highlights:
    enabled: true
    color: "#FFFF0055"

graphic_elements:
  arrows: true
  zoom_effects: true
  blur_sensitive: true

output:
  formats: ["jpg", "png", "pdf"]
  resolutions: ["web", "print", "mobile"]
```

2. Основной скрипт обработки:

python

```python
import yaml
from PIL import Image
import os

class ScreenshotProcessor:
    """Основной класс для обработки скриншотов"""

    def __init__(self, config_path):
        with open(config_path, 'r') as f:
            self.config = yaml.safe_load(f)

        self.text_annotator = TextAnnotation()
        self.graphic_annotator = GraphicAnnotations()

    def process_screenshot(self, input_path, output_dir):
        """Обработка одного скриншота"""
        original = Image.open(input_path)

        # Применение цветовой оптимизации
        color_optimized = self.optimize_colors(original)

        # Добавление аннотаций
        annotated = self.add_annotations(color_optimized)

        # Экспорт в различные форматы
        self.export_formats(annotated, output_dir, os.path.basename(input_path))

    def batch_process(self, input_dir, output_dir):
        """Пакетная обработка всех скриншотов"""
        for filename in os.listdir(input_dir):
            if filename.lower().endswith(('.png', '.jpg', '.jpeg')):
                input_path = os.path.join(input_dir, filename)
                self.process_screenshot(input_path, output_dir)
```

Итоговое задание для самостоятельного выполнения:

Сценарий: Подготовка комплекта аннотированных скриншотов для руководства пользователя корпоративного ПО.

Задание:

1. Подготовка исходных материалов:

- 10 скриншотов ключевых функций приложения
- Включить интерфейсы с формами, таблицами, меню
- Захватить различные состояния интерфейса

2. Оптимизация цветности:

python

```python
# Создание различных цветовых версий
color_versions = {
    'web_srgb': {'space': 'sRGB', 'optimization': 'web'},
    'print_cmyk': {'space': 'CMYK', 'optimization': 'print'},
    'ebook_grayscale': {'space': 'L', 'optimization': 'ebook'},
    'mobile_p3': {'space': 'RGB', 'optimization': 'mobile'}
}
```

3. Создание аннотаций:

markdown

```markdown
### Типы аннотаций для применения:
1. **Нумерация шагов** - для пошаговых инструкций
2. **Текстовые выноски** - пояснение элементов
3. **Выделение областей** - акцентирование внимания
4. **Стрелки-указатели** - направление взгляда
5. **Эффекты увеличения** - детализация мелких элементов
6. **Размытие** - скрытие конфиденциальной информации
```

4. Структура проекта:

text

```
professional_screenshot_guide/
├── 01_raw_screenshots/
├── 02_color_optimized/
│   ├── web_srgb/
│   ├── print_cmyk/
│   ├── ebook_grayscale/
│   └── mobile_p3/
├── 03_annotated_versions/
│   ├── step_by_step/
│   ├── feature_highlights/
│   ├── interface_guide/
│   └── troubleshooting/
├── 04_final_outputs/
│   ├── pdf_manual/
│   ├── web_gallery/
│   ├── print_ready/
│   └── mobile_assets/
├── scripts/
│   ├── color_optimization.py
│   ├── annotation_tools.py
│   └── batch_processor.py
└── documentation/
    ├── color_profiles.md
    ├── annotation_guide.md
    └── quality_report.md
```

5. Требования к результату:

Качество цветопередачи:

- Корректное отображение на целевых устройствах
- Сохранение читаемости текста и интерфейсов
- Единообразие цветового решения

Профессиональные аннотации:

- Четкая иерархия информации
- Единый стиль оформления
- Информативность и лаконичность

Технические требования:

- Оптимизированные размеры файлов
- Поддержка необходимых форматов
- Корректные цветовые профили

6. Критерии оценки:

Компонент	Критерии
Качество цветопередачи	Корректность профилей, адаптация к носител
Профессионализм аннотаций	Ясность, единый стиль, информативность

Компонент	Критерии
Техническая реализация	Оптимизация, форматы, организация
Креативность и ясность	Эффективность коммуникации, дизайн

Это занятие даст студентам профессиональные навыки подготовки высококачественных иллюстративных материалов, которые эффективно коммуницируют информацию и корректно отображаются на любых устройствах.
Практическое занятие 63 - 66. Опрос экспертов и анализ по-лученных сведений
Отлично! Разработаем комплексный план практического занятия по проведению опросов экспертов и анализу полученных данных.

Практическое занятие 63-66: Опрос экспертов и анализ полученных сведений
Цель занятия: Освоить методы планирования, проведения и анализа экспертных опросов для сбора качественной информации в профессиональной деятельности.
Программное обеспечение:

- **Опросы:** Google Forms, Microsoft Forms, Typeform
- **Анализ:** Excel, SPSS, R, Python (pandas, matplotlib)
- **Организация:** Trello, Notion, Miro
- **Коммуникация:** Zoom, Microsoft Teams, Calendly
- **Качественный анализ:** NVivo, MAXQDA, Atlas.ti

Необходимые знания: Базовые навыки исследования, понимание принципов работы с данными.

План занятия и Ход работы
Часть 1: Подготовка экспертного опроса (≈4 часа)
Задача: Научиться планировать и разрабатывать методологию экспертного опроса.
1. Этапы подготовки опроса:
2. Разработка методологии исследования:
python

```
class ResearchMethodology:
    """Класс для разработки методологии исследования"""

    def __init__(self, research_topic):
        self.topic = research_topic
        self.methods = []
        self.timeline = {}

    def add_method(self, method, description, tools):
        """Добавление метода исследования"""
        self.methods.append({
            'method': method,
            'description': description,
            'tools': tools
        })

    def create_questionnaire_template(self):
        """Шаблон анкеты для экспертного опроса"""
        template = {
            "section_1": {
                "title": "Профессиональный опыт",
                "questions": [
                    {
                        "type": "multiple_choice",
                        "question": "Стаж работы в области",
                        "options": ["1-3 года", "3-5 лет", "5-10 лет", "более 10 лет"]
                    }
                ]
            },
            "section_2": {
                "title": "Оценка проблемных областей",
                "questions": [
                    {
                        "type": "likert_scale",
                        "question": "Оцените актуальность проблемы",
                        "scale": ["1 - Не актуально", "2", "3", "4", "5 - Очень актуально"]
                    }
                ]
            }
        }
        return template
```

3. Практическое задание "Разработка инструментария":
markdown

Задание: Создание анкеты для экспертов по теме
"Перспективы развития искусственного интеллекта в образовании"

Структура анкеты:
1. **Блок A: Демографические данные**
   - Сфера деятельности
   - Опыт работы
   - Уровень экспертизы

2. **Блок B: Оценка текущего состояния**
   - Актуальные технологии
   - Проблемы внедрения
   - Эффективность решений

3. **Блок C: Прогноз развития**
   - Перспективные направления
   - Ожидаемые сроки
   - Факторы влияния

4. **Блок D: Рекомендации**
   - Приоритетные действия
   - Ресурсные потребности
   - Ограничения

Часть 2: Проведение экспертного опроса (≈4 часа)
Задача: Освоить техники проведения интервью и организации фокус-групп.
1. Методы сбора экспертных данных:
python
```
class ExpertInterview:
    """Класс для проведения экспертных интервью"""

    def __init__(self, expert_profile):
        self.expert = expert_profile
        self.questions = []
        self.recording_consent = False

    def prepare_interview_guide(self):
        """Подготовка гайда интервью"""
        guide = {
            "introduction": {
                "purpose": "Объяснение целей исследования",
                "duration": "Информирование о времени",
                "confidentiality": "Гарантии конфиденциальности"
            },
            "main_questions": [
                {
                    "type": "open_ended",
                    "question": "Какие основные тенденции вы наблюдаете?",
                    "probes": ["Можете привести пример?", "Что это означает на практике?"]
                }
            ],
            "conclusion": {
                "summary": "Резюме ключевых моментов",
                "additional_info": "Возможность добавить информацию",
                "next_steps": "Планы по использованию результатов"
            }
        }
        return guide

    def conduct_remote_interview(self, platform="Zoom"):
        """Проведение удаленного интервью"""
        checklist = [
            "Проверить оборудование и связь",
            "Подготовить материалы для демонстрации",
            "Записать согласие на запись",
            "Следовать таймингу",
            "Активно слушать и задавать уточняющие вопросы"
        ]
        return checklist
```
2. Практическое задание "Проведение фокус-группы":
markdown
Сценарий фокус-группы с IT-экспертами

****Тема:**** "Будущее облачных технологий в корпоративном секторе"

****Модераторские техники:****

- Техника лестницы (Laddering) - углубление в причины
- Техника проекций - выявление скрытых установок
- Техника ассоциаций - выявление глубинных связей

****Вопросы для обсуждения:****
1. Какие преимущества облачных технологий наиболее значимы?
2. Какие риски препятствуют массовому внедрению?
3. Как изменится ландшафт через 3-5 лет?

****Анализ during-сессии:****
- Выявление консенсусных позиций
- Фиксация конфликтующих мнений
- Отслеживание невербальных реакций

Часть 3: Качественный анализ данных (≈4 часа)
Задача: Научиться анализировать качественные данные и выявлять закономерности.
1. Методы качественного анализа:
python
```
import pandas as pd
from collections import Counter
import re

class QualitativeAnalyzer:
    """Анализ качественных данных из экспертных опросов"""

    def __init__(self, interview_transcripts):
        self.transcripts = interview_transcripts
        self.themes = []
        self.codes = {}

    def thematic_analysis(self):
        """Тематический анализ интервью"""
        # Этапы тематического анализа по Брауну и Кларку
        stages = [
            "Знакомство с данными",
            "Генерация первоначальных кодов",
            "Поиск тем",
            "Пересмотр тем",
            "Определение и именование тем",
            "Подготовка отчета"
        ]

        themes_found = self.identify_themes()
        return themes_found

    def content_analysis(self, text_data):
        """Контент-анализ текстовых данных"""
        # Анализ частотности терминов
        words = re.findall(r'\b\w+\b', text_data.lower())
        word_freq = Counter(words)

        # Исключение стоп-слов
        stop_words = {'и', 'в', 'на', 'с', 'по', 'о', 'это', 'как', 'для'}
        meaningful_words = {word: count for word, count in word_freq.items()
                            if word not in stop_words and len(word) > 3}

        return dict(meaningful_words.most_common(10))

    def code_transcripts(self, codebook):
        """Кодирование транскриптов по кодбуку"""
        coded_data = {}

        for theme, keywords in codebook.items():
            coded_data[theme] = {}
            for keyword in keywords:
                count = sum(1 for transcript in self.transcripts
                            if keyword.lower() in transcript.lower())
                coded_data[theme][keyword] = count

        return coded_data
```
2. Практическое задание "Анализ экспертных интервью":
python
```
# Пример анализа транскрипта интервью
def analyze_expert_transcript(transcript):
    """Анализ одного экспертного интервью"""
```

```
    analyzer = QualitativeAnalyzer([transcript])

    # Кодбук для анализа
    codebook = {
        "технологии": ["искусственный интеллект", "машинное обучение", "нейросети"],
        "проблемы": ["внедрение", "стоимость", "квалификация"],
        "перспективы": ["будущее", "развитие", "потенциал"]
    }

    # Кодирование данных
    coded_results = analyzer.code_transcripts(codebook)

    # Контент-анализ
    content_analysis = analyzer.content_analysis(transcript)

    return {
        'coded_data': coded_results,
        'frequent_terms': content_analysis,
        'main_themes': analyzer.thematic_analysis()
    }

# Применение
transcript = """
Эксперт обсуждает перспективы искусственного интеллекта в образовании.
Внедрение AI сталкивается с проблемами стоимости и нехватки квалификации.
Однако будущее развитие показывает большой потенциал машинного обучения.
"""

results = analyze_expert_transcript(transcript)
print(results)
```

Часть 4: Количественный анализ и визуализация (≈4 часа)
Задача: Научиться обрабатывать количественные данные из опросов и создавать наглядные отчеты.
1. Анализ данных опроса:

```
python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats

class SurveyAnalyzer:
    """Анализ данных экспертного опроса"""

    def __init__(self, survey_data):
        self.df = pd.DataFrame(survey_data)
        self.analysis_results = {}

    def descriptive_analysis(self):
        """Дескриптивный анализ данных"""
        desc_stats = {
            'mean_scores': self.df.select_dtypes(include=['number']).mean(),
            'std_scores': self.df.select_dtypes(include=['number']).std(),
            'response_counts': self.df.count(),
            'missing_data': self.df.isnull().sum()
        }
        return desc_stats

    def correlation_analysis(self):
        """Корреляционный анализ"""
        numeric_data = self.df.select_dtypes(include=['number'])
        correlation_matrix = numeric_data.corr()
        return correlation_matrix

    def visualize_results(self):
        """Визуализация результатов опроса"""
        fig, axes = plt.subplots(2, 2, figsize=(15, 10))

        # График 1: Распределение оценок
        self.df.hist(ax=axes[0, 0])
        axes[0, 0].set_title('Распределение оценок экспертов')

        # График 2: Тепловая карта корреляций
        sns.heatmap(self.correlation_analysis(), annot=True, ax=axes[0, 1])
        axes[0, 1].set_title('Корреляция между переменными')

        # График 3: Боксплот оценок
```

```
        self.df.boxplot(ax=axes[1, 0])
        axes[1, 0].set_title('Боксплот оценок')

        plt.tight_layout()
        return fig

    def export_analysis_report(self, filename):
        """Экспорт отчета анализа"""
        report = {
            'descriptive_statistics': self.descriptive_analysis(),
            'correlation_analysis': self.correlation_analysis(),
            'data_quality': {
                'completion_rate': (self.df.count() / len(self.df)) * 100,
                'missing_values': self.df.isnull().sum().sum()
            }
        }

        # Сохранение в различных форматах
        self.df.to_csv(f"{filename}.csv", index=False)
        with open(f"{filename}_report.json", 'w') as f:
            json.dump(report, f, indent=2, ensure_ascii=False)

        return report
```

2. Практическое задание "Анализ данных опроса":

```
python
# Пример данных опроса
survey_data = {
    'expert_id': [1, 2, 3, 4, 5],
    'experience_years': [10, 5, 8, 3, 12],
    'trend_importance_ai': [5, 4, 5, 3, 5],
    'implementation_difficulty': [4, 3, 5, 2, 4],
    'future_potential': [5, 4, 5, 4, 5]
}

# Анализ данных
analyzer = SurveyAnalyzer(survey_data)

# Получение результатов
stats = analyzer.descriptive_analysis()
correlations = analyzer.correlation_analysis()

print("Средние оценки:", stats['mean_scores'])
print("Корреляции:", correlations)

# Визуализация
analyzer.visualize_results()
plt.show()
```

Часть 5: Подготовка итогового отчета (≈4 часа)

Задача: Научиться синтезировать результаты и готовить профессиональные отчеты.

1. Структура экспертного отчета:

```
python
class ExpertReport:
    """Класс для подготовки отчета по результатам экспертного опроса"""

    def __init__(self, research_topic, methodology, findings):
        self.topic = research_topic
        self.methodology = methodology
        self.findings = findings
        self.recommendations = []

    def create_executive_summary(self):
        """Создание исполнительного резюме"""
        summary = {
            "research_objective": f"Анализ {self.topic}",
            "key_findings": self.extract_key_insights(),
            "main_recommendations": self.recommendations[:3],
            "methodology_brief": f"Опрос {self.methodology['expert_count']} экспертов"
        }
        return summary

    def generate_full_report(self):
        """Генерация полного отчета"""
        report_structure = {
            "Титульная страница": {
                "Название исследования": self.topic,
```

```
            "Дата проведения": "2024",
            "Исполнитель": "Исследовательская группа"
        },
        "Содержание": "Автоматически генерируемое оглавление",
        "Введение": {
            "Актуальность темы": "Обоснование исследования",
            "Цели и задачи": "Конкретные цели исследования",
            "Методология": self.methodology
        },
        "Результаты": {
            "Качественный анализ": self.findings['qualitative'],
            "Количественный анализ": self.findings['quantitative'],
            "Визуализация данных": "Графики и диаграммы"
        },
        "Выводы и рекомендации": {
            "Ключевые выводы": self.extract_key_insights(),
            "Рекомендации": self.recommendations,
            "Перспективы дальнейших исследований": "Направления развития"
        },
        "Приложения": {
            "Анкета опроса": "Полный текст анкеты",
            "Профили экспертов": "Демография участников",
            "Статистические таблицы": "Детальные данные"
        }
    }
    return report_structure
```

2. Практическое задание "Подготовка финального отчета":

markdown

```
### Отчет по экспертному опросу
"Стратегические перспективы развития AI в здравоохранении"

#### Резюме для руководства:
- **Основной вывод:** 80% экспертов считают AI ключевым драйвером
- **Критические риски:** Отсутствие регулирования и этические вопросы
- **Рекомендации:** Разработать стандарты и увеличить инвестиции в R&D

#### Детальные разделы:

1. **Методология исследования**
   - Выборка: 25 экспертов из 5 стран
   - Методы: Глубинные интервью, онлайн-опрос, фокус-группы
   - Период: Январь-Март 2024

2. **Ключевые находки**
   - Единство в оценке потенциала (4.8/5.0)
   - Разногласия в сроках внедрения (2-10 лет)
   - Консенсус по необходимым инвестициям

3. **Визуализации**
   - Радарные диаграммы компетенций
   - Heat map приоритетных направлений
   - Roadmap внедрения технологий
```

Итоговое задание для самостоятельного выполнения:

Сценарий: Проведение полного цикла экспертного опроса по актуальной технологической теме.

Задание:

1. Подготовительный этап:

python

```
# Выбор темы исследования
research_topic = "Влияние больших данных на принятие бизнес-решений"

# Разработка методологии
methodology = {
    "research_type": "Смешанные методы",
    "data_collection": ["Онлайн-опрос", "Экспертные интервью"],
    "sample_size": 15,
    "expert_criteria": ["Опыт > 5 лет", "Публикации", "Практический опыт"]
}
```

2. Проведение исследования:

- Разработка анкеты (15-20 вопросов)
- Подбор и приглашение экспертов
- Проведение 3-5 глубинных интервью
- Организация фокус-группы

3. Анализ данных:

python

```
# Комплексный анализ
```

```
def comprehensive_analysis(qualitative_data, quantitative_data):
    """Комплексный анализ качественных и количественных данных"""

    # Триангуляция данных
    triangulation_results = {
        'converging_evidence': find_common_themes(qualitative_data, quantitative_data),
        'diverging_points': identify_contradictions(qualitative_data, quantitative_data),
        'data_completeness': assess_data_quality(qualitative_data, quantitative_data)
    }

    return triangulation_results
```

4. Структура проекта:

```
text
expert_survey_project/
├── 01_research_design/
│   ├── methodology.md
│   ├── questionnaire.pdf
│   └── expert_database.csv
├── 02_data_collection/
│   ├── survey_responses/
│   ├── interview_transcripts/
│   ├── focus_group_recording/
│   └── consent_forms/
├── 03_analysis/
│   ├── qualitative_analysis/
│   │   ├── coded_transcripts/
│   │   ├── theme_maps/
│   │   └── narrative_summaries/
│   ├── quantitative_analysis/
│   │   ├── descriptive_stats/
│   │   ├── correlation_analysis/
│   │   └── visualization/
│   └── mixed_methods_integration/
├── 04_reporting/
│   ├── executive_summary.pdf
│   ├── full_report.docx
│   ├── presentation_slides.pptx
│   └── infographics/
└── 05_supplementary/
    ├── literature_review/
    ├── ethical_considerations/
    └── future_research_directions/
```

5. Требования к результату:

Методологическая строгость:

- Обоснование выборки экспертов
- Валидность инструментария
- Этические стандарты исследования

Глубина анализа:

- Интеграция качественных и количественных данных
- Выявление скрытых закономерностей
- Контекстуализация результатов

Практическая значимость:

- Конкретные рекомендации для отрасли
- Roadmap внедрения результатов
- Оценка экономического эффекта

6. Критерии оценки:

Компонент	Критерии
Методология	Обоснованность подхода, качество инструментария
Качество данных	Репрезентативность выборки, глубина интервью
Анализ	Комплексность, инновационность подходов
Отчетность	Структура, ясность, практическая ценность
Этика и профессионализм	Соблюдение стандартов, оформление

Это занятие даст студентам комплексные навыки проведения экспертных исследований - от планирования до представления результатов, что является критически важной компетенцией для аналитиков, исследователей и консультантов.
Практическое занятие 67 - 70. Исследование программных средства на тестовом стенде
Отлично! Разработаем комплексный план практического занятия по исследованию программных средств на тестовом стенде.

Практическое занятие 67-70: Исследование программных средств на тестовом стенде
Цель занятия: Освоить методы тестирования и исследования программного обеспечения в контролируемой среде, включая функциональное тестирование, нагрузочное тестирование и анализ безопасности.
Программное обеспечение и оборудование:

- **Виртуализация:** VMware, VirtualBox, Docker
- **Мониторинг:** Grafana, Prometheus, Wireshark
- **Тестирование:** Selenium, JMeter, Postman
- **Анализ кода:** SonarQube, ESLint, Pylint
- **Безопасность:** OWASP ZAP, Nessus, Metasploit
- **Документирование:** Jupyter Notebook, Confluence, DokuWiki

Необходимые знания: Базовые навыки программирования, понимание принципов работы ПО, основы сетевых технологий.

План занятия и Ход работы
Часть 1: Подготовка тестового стенда (≈4 часа)
Задача: Научиться развертывать и настраивать изолированную тестовую среду.
1. Архитектура тестового стенда:
2. Автоматизация развертывания:
python

```
#!/usr/bin/env python3
"""
Скрипт автоматизации развертывания тестового стенда
"""

import docker
import subprocess
import time
import yaml

class TestLabDeployer:
    def __init__(self, config_path='lab_config.yaml'):
        self.client = docker.from_env()
        self.config = self.load_config(config_path)

    def load_config(self, config_path):
        """Загрузка конфигурации стенда"""
        with open(config_path, 'r') as f:
            return yaml.safe_load(f)

    def deploy_web_server(self):
        """Развертывание веб-сервера"""
        print("Развертывание nginx...")
        nginx_config = {
            'image': 'nginx:latest',
            'ports': {'80/tcp': 8080},
            'name': 'test-webserver',
            'detach': True
        }
        return self.client.containers.run(**nginx_config)

    def deploy_database(self):
        """Развертывание базы данных"""
        print("Развертывание PostgreSQL...")
        db_config = {
            'image': 'postgres:13',
            'environment': {
                'POSTGRES_DB': 'test_db',
                'POSTGRES_USER': 'tester',
                'POSTGRES_PASSWORD': 'test123'
            },
            'ports': {'5432/tcp': 5432},
            'name': 'test-database',
            'detach': True
        }
        return self.client.containers.run(**db_config)

    def deploy_monitoring(self):
        """Развертывание системы мониторинга"""
        print("Настройка Prometheus + Grafana...")
```

```
        # Docker Compose для мониторинга
        compose_cmd = [
            'docker-compose', '-f', 'monitoring/docker-compose.yml', 'up', '-d'
        ]
        subprocess.run(compose_cmd, check=True)

    def full_deployment(self):
        """Полное развертывание стенда"""
        print("=== Начало развертывания тестового стенда ===")

        services = []
        services.append(self.deploy_web_server())
        time.sleep(5)

        services.append(self.deploy_database())
        time.sleep(10)

        self.deploy_monitoring()

        print("=== Тестовый стенд готов к работе ===")
        return services

# Конфигурационный файл lab_config.yaml
config_example = """
lab:
  name: "performance_test_lab"
  version: "1.0"

network:
  subnet: "172.20.0.0/16"
  gateway: "172.20.0.1"

services:
  webserver:
    image: "nginx:latest"
    ports:
      - "8080:80"

  database:
    image: "postgres:13"
    environment:
      POSTGRES_DB: "test_app"
      POSTGRES_USER: "test_user"

monitoring:
  prometheus_port: 9090
  grafana_port: 3000
"""
```

3. Практическое задание "Создание стенда":

```
bash
#!/bin/bash
# setup_test_lab.sh - Скрипт настройки тестового стенда

echo "Инициализация тестового стенда..."

# Создание виртуальной сети
docker network create --subnet=172.20.0.0/16 test-lab

# Запуск контейнеров
docker run -d --name test-web --network test-lab -p 80:80 nginx
docker run -d --name test-db --network test-lab -e MYSQL_ROOT_PASSWORD=test123 mysql:8.0
docker run -d --name test-monitor --network test-lab -p 3000:3000 grafana/grafana

echo "Проверка состояния сервисов..."
docker ps -a

echo "Тестовый стенд готов!"
```

Часть 2: Функциональное тестирование (≈4 часа)

Задача: Освоить методы функционального тестирования веб-приложений и API.

1. Автоматизация тестирования с Selenium:

```
python
import unittest
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
```

```
from selenium.webdriver.support import expected_conditions as EC
import time

class FunctionalTests(unittest.TestCase):
    """Класс для функционального тестирования веб-приложения"""

    def setUp(self):
        """Настройка перед каждым тестом"""
        self.driver = webdriver.Chrome()
        self.driver.implicitly_wait(10)
        self.base_url = "http://localhost:8080"
        self.wait = WebDriverWait(self.driver, 15)

    def test_user_login(self):
        """Тест авторизации пользователя"""
        driver = self.driver
        driver.get(self.base_url + "/login")

        # Ввод учетных данных
        username_field = self.wait.until(
            EC.presence_of_element_located((By.ID, "username"))
        )
        username_field.send_keys("testuser")

        password_field = driver.find_element(By.ID, "password")
        password_field.send_keys("testpass123")

        # Нажатие кнопки входа
        login_button = driver.find_element(By.XPATH, "//button[@type='submit']")
        login_button.click()

        # Проверка успешного входа
        welcome_message = self.wait.until(
            EC.presence_of_element_located((By.CLASS_NAME, "welcome-message"))
        )
        self.assertIn("Добро пожаловать", welcome_message.text)

    def test_form_validation(self):
        """Тест валидации форм"""
        driver = self.driver
        driver.get(self.base_url + "/contact")

        # Попытка отправки пустой формы
        submit_button = driver.find_element(By.ID, "submit-btn")
        submit_button.click()

        # Проверка сообщений об ошибках
        error_messages = driver.find_elements(By.CLASS_NAME, "error-message")
        self.assertTrue(len(error_messages) > 0)

    def test_navigation(self):
        """Тест навигации по сайту"""
        driver = self.driver
        driver.get(self.base_url)

        # Проверка доступности основных страниц
        pages_to_test = ["/about", "/services", "/contact", "/products"]

        for page in pages_to_test:
            with self.subTest(page=page):
                driver.get(self.base_url + page)
                self.assertNotIn("404", driver.title)
                self.assertTrue(driver.find_element(By.TAG_NAME, "body").is_displayed())

    def tearDown(self):
        """Завершение после каждого теста"""
        self.driver.quit()

if __name__ == "__main__":
    # Запуск тестов с генерацией отчета
    import xmlrunner
    unittest.main(
        testRunner=xmlrunner.XMLTestRunner(output='test-reports'),
        failfast=False, buffer=False, catchbreak=False
    )
```

2. Тестирование API с Postman и Python:

python

```
import requests
import json
import pytest

class APITestSuite:
    """Набор тестов для REST API"""

    BASE_URL = "http://localhost:3000/api"

    def test_health_check(self):
        """Тест доступности API"""
        response = requests.get(f"{self.BASE_URL}/health")
        assert response.status_code == 200
        assert response.json()["status"] == "OK"

    def test_user_crud_operations(self):
        """Тест CRUD операций для пользователей"""
        # CREATE
        user_data = {
            "name": "Test User",
            "email": "test@example.com",
            "password": "securepassword"
        }
        create_response = requests.post(f"{self.BASE_URL}/users", json=user_data)
        assert create_response.status_code == 201
        user_id = create_response.json()["id"]

        # READ
        get_response = requests.get(f"{self.BASE_URL}/users/{user_id}")
        assert get_response.status_code == 200
        assert get_response.json()["email"] == user_data["email"]

        # UPDATE
        update_data = {"name": "Updated Test User"}
        update_response = requests.put(f"{self.BASE_URL}/users/{user_id}", json=update_data)
        assert update_response.status_code == 200

        # DELETE
        delete_response = requests.delete(f"{self.BASE_URL}/users/{user_id}")
        assert delete_response.status_code == 204

        # Verify deletion
        verify_response = requests.get(f"{self.BASE_URL}/users/{user_id}")
        assert verify_response.status_code == 404

    def test_api_performance(self):
        """Тест производительности API"""
        import time

        start_time = time.time()
        responses = []

        # 100 последовательных запросов
        for i in range(100):
            response = requests.get(f"{self.BASE_URL}/products")
            responses.append(response.status_code)

        end_time = time.time()
        total_time = end_time - start_time

        # Все запросы должны быть успешными
        assert all(code == 200 for code in responses)
        # Время выполнения не должно превышать 10 секунд
        assert total_time < 10.0

        print(f"API performance: {total_time:.2f} seconds for 100 requests")
        return total_time

# Запуск тестов
if __name__ == "__main__":
    test_suite = APITestSuite()

    try:
        test_suite.test_health_check()
        test_suite.test_user_crud_operations()
        performance_time = test_suite.test_api_performance()
```

```
        print("Все тесты прошли успешно!")
        print(f"Среднее время ответа: {performance_time/100:.3f} сек")

    except AssertionError as e:
        print(f"Тест не пройден: {e}")
    except Exception as e:
        print(f"Ошибка при выполнении тестов: {e}")
```

Часть 3: Нагрузочное тестирование (≈4 часа)
Задача: Освоить методы тестирования производительности и нагрузки.
1. Нагрузочное тестирование с JMeter:

```
python
import subprocess
import time
import xml.etree.ElementTree as ET
from datetime import datetime

class PerformanceTester:
    """Класс для управления нагрузочным тестированием"""

    def __init__(self, jmeter_path="/opt/jmeter/bin/jmeter"):
        self.jmeter_path = jmeter_path
        self.test_plans = {}

    def create_simple_test_plan(self, test_name, target_url, users=100, duration=300):
        """Создание простого тестового плана JMeter"""
        jmx_template = f"""
<?xml version="1.0" encoding="UTF-8"?>
<jmeterTestPlan version="1.2" properties="5.0" jmeter="5.5">
  <hashTree>
    <TestPlan guiclass="TestPlanGui" testclass="TestPlan" testname="{test_name}">
      <stringProp name="TestPlan.comments"></stringProp>
      <boolProp name="TestPlan.functional_mode">false</boolProp>
      <boolProp name="TestPlan.tearDown_on_shutdown">true</boolProp>
      <boolProp name="TestPlan.serialize_threadgroups">false</boolProp>
      <elementProp name="TestPlan.user_defined_variables" elementType="Arguments" guiclass="ArgumentsPanel"
testclass="Arguments" testname="User Defined Variables">
        <collectionProp name="Arguments.arguments"/>
      </elementProp>
    </TestPlan>
    <hashTree>
      <ThreadGroup guiclass="ThreadGroupGui" testclass="ThreadGroup" testname="Thread Group">
        <stringProp name="ThreadGroup.on_sample_error">continue</stringProp>
        <elementProp name="ThreadGroup.main_controller" elementType="LoopController" guiclass="LoopControlPanel"
testclass="LoopController" testname="Loop Controller">
          <boolProp name="LoopController.continue_forever">false</boolProp>
          <stringProp name="LoopController.loops">-1</stringProp>
        </elementProp>
        <stringProp name="ThreadGroup.num_threads">{users}</stringProp>
        <stringProp name="ThreadGroup.ramp_time">60</stringProp>
        <boolProp name="ThreadGroup.scheduler">true</boolProp>
        <stringProp name="ThreadGroup.duration">{duration}</stringProp>
        <stringProp name="ThreadGroup.delay">0</stringProp>
      </ThreadGroup>
      <hashTree>
        <HTTPSamplerProxy guiclass="HttpTestSampleGui" testclass="HTTPSamplerProxy" testname="HTTP Request">
          <boolProp name="HTTPSampler.postBodyRaw">false</boolProp>
          <elementProp name="HTTPsampler.Arguments" elementType="Arguments">
            <collectionProp name="Arguments.arguments"/>
          </elementProp>
          <stringProp name="HTTPSampler.domain">localhost</stringProp>
          <stringProp name="HTTPSampler.port">8080</stringProp>
          <stringProp name="HTTPSampler.protocol">http</stringProp>
          <stringProp name="HTTPSampler.contentEncoding"></stringProp>
          <stringProp name="HTTPSampler.path">/</stringProp>
          <stringProp name="HTTPSampler.method">GET</stringProp>
          <boolProp name="HTTPSampler.follow_redirects">true</boolProp>
          <boolProp name="HTTPSampler.auto_redirects">false</boolProp>
          <boolProp name="HTTPSampler.use_keepalive">true</boolProp>
          <boolProp name="HTTPSampler.DO_MULTIPART_POST">false</boolProp>
          <stringProp name="HTTPSampler.embedded_url_re"></stringProp>
          <stringProp name="HTTPSampler.connect_timeout"></stringProp>
          <stringProp name="HTTPSampler.response_timeout"></stringProp>
        </HTTPSamplerProxy>
        <hashTree/>
      </hashTree>
```

```
    </hashTree>
  </hashTree>
</jmeterTestPlan>
"""

        filename = f"test_plans/{test_name}.jmx"
        with open(filename, 'w') as f:
            f.write(jmx_template)

        self.test_plans[test_name] = filename
        return filename

    def run_load_test(self, test_plan_name, results_file=None):
        """Запуск нагрузочного теста"""
        if results_file is None:
            results_file = f"results/{test_plan_name}_{datetime.now().strftime('%Y%m%d_%H%M%S')}.jtl"

        jmx_file = self.test_plans[test_plan_name]

        cmd = [
            self.jmeter_path,
            '-n',  # non-GUI mode
            '-t', jmx_file,
            '-l', results_file,
            '-e',  # generate report dashboard
            '-o', f"reports/{test_plan_name}_report"
        ]

        print(f"Запуск нагрузочного теста: {test_plan_name}")
        start_time = time.time()

        try:
            result = subprocess.run(cmd, check=True, capture_output=True, text=True)
            end_time = time.time()

            print(f"Тест завершен за {end_time - start_time:.2f} секунд")
            print(f"Результаты сохранены в: {results_file}")

            return {
                'success': True,
                'duration': end_time - start_time,
                'results_file': results_file
            }

        except subprocess.CalledProcessError as e:
            print(f"Ошибка при выполнении теста: {e}")
            return {'success': False, 'error': str(e)}

    def analyze_jmeter_results(self, results_file):
        """Анализ результатов JMeter"""
        tree = ET.parse(results_file)
        root = tree.getroot()

        statistics = {
            'total_requests': 0,
            'successful_requests': 0,
            'failed_requests': 0,
            'average_response_time': 0,
            'min_response_time': float('inf'),
            'max_response_time': 0,
            'throughput': 0
        }

        for sample in root.findall('.//httpSample'):
            statistics['total_requests'] += 1

            success = sample.get('s') == 'true'
            if success:
                statistics['successful_requests'] += 1
            else:
                statistics['failed_requests'] += 1

            response_time = int(sample.get('t', 0))
            statistics['average_response_time'] += response_time

            if response_time < statistics['min_response_time']:
                statistics['min_response_time'] = response_time
```

```
            if response_time > statistics['max_response_time']:
                statistics['max_response_time'] = response_time

        if statistics['total_requests'] > 0:
            statistics['average_response_time'] /= statistics['total_requests']
            statistics['throughput'] = statistics['total_requests'] / statistics.get('test_duration', 300)

        return statistics

# Использование
tester = PerformanceTester()
tester.create_simple_test_plan("website_load_test", "http://localhost:8080", users=50, duration=600)
results = tester.run_load_test("website_load_test")

if results['success']:
    stats = tester.analyze_jmeter_results(results['results_file'])
    print("Результаты нагрузочного тестирования:")
    for key, value in stats.items():
        print(f"  {key}: {value}")
```

2. Мониторинг ресурсов во время тестирования:

```
python
import psutil
import time
import pandas as pd
from threading import Thread

class ResourceMonitor:
    """Мониторинг системных ресурсов во время тестирования"""

    def __init__(self, interval=1):
        self.interval = interval
        self.monitoring = False
        self.data = []

    def start_monitoring(self):
        """Запуск мониторинга"""
        self.monitoring = True
        self.monitor_thread = Thread(target=self._monitor_loop)
        self.monitor_thread.start()
        print("Мониторинг ресурсов запущен...")

    def _monitor_loop(self):
        """Цикл сбора метрик"""
        while self.monitoring:
            timestamp = time.time()

            # Сбор метрик
            cpu_percent = psutil.cpu_percent(interval=None)
            memory = psutil.virtual_memory()
            disk_io = psutil.disk_io_counters()
            network_io = psutil.net_io_counters()

            metrics = {
                'timestamp': timestamp,
                'cpu_percent': cpu_percent,
                'memory_percent': memory.percent,
                'memory_used_gb': memory.used / (1024**3),
                'disk_read_mb': disk_io.read_bytes / (1024**2) if disk_io else 0,
                'disk_write_mb': disk_io.write_bytes / (1024**2) if disk_io else 0,
                'network_sent_mb': network_io.bytes_sent / (1024**2),
                'network_recv_mb': network_io.bytes_recv / (1024**2)
            }

            self.data.append(metrics)
            time.sleep(self.interval)

    def stop_monitoring(self):
        """Остановка мониторинга"""
        self.monitoring = False
        self.monitor_thread.join()
        print("Мониторинг ресурсов остановлен")

    def generate_report(self):
        """Генерация отчета по мониторингу"""
        if not self.data:
            return None
```

```
        df = pd.DataFrame(self.data)
        report = {
            'duration_seconds': df['timestamp'].max() - df['timestamp'].min(),
            'avg_cpu_usage': df['cpu_percent'].mean(),
            'max_cpu_usage': df['cpu_percent'].max(),
            'avg_memory_usage': df['memory_percent'].mean(),
            'max_memory_usage': df['memory_percent'].max(),
            'total_disk_read_mb': df['disk_read_mb'].sum(),
            'total_disk_write_mb': df['disk_write_mb'].sum(),
            'total_network_sent_mb': df['network_sent_mb'].sum(),
            'total_network_recv_mb': df['network_recv_mb'].sum()
        }

        return report

# Использование мониторинга
monitor = ResourceMonitor(interval=2)

print("Запуск мониторинга во время нагрузочного теста...")
monitor.start_monitoring()

# Здесь выполняется нагрузочный тест
time.sleep(30)  # Имитация работы теста

monitor.stop_monitoring()
report = monitor.generate_report()

print("Отчет по использованию ресурсов:")
for metric, value in report.items():
    print(f"{metric}: {value:.2f}")
```

Часть 4: Анализ безопасности (≈4 часа)
Задача: Освоить базовые методы тестирования безопасности приложений.
1. Сканирование уязвимостей с OWASP ZAP:
python
```
from zapv2 import ZAPv2
import time

class SecurityTester:
    """Класс для тестирования безопасности веб-приложений"""

    def __init__(self, zap_proxy='http://localhost:8080', api_key=None):
        self.zap = ZAPv2(apikey=api_key, proxies={'http': zap_proxy, 'https': zap_proxy})

    def passive_scan(self, target_url):
        """Пассивное сканирование безопасности"""
        print(f"Запуск пассивного сканирования для {target_url}")

        # Начало пассивного сканирования
        self.zap.urlopen(target_url)
        time.sleep(5)  # Ожидание загрузки страницы

        # Проверка прогресса сканирования
        while int(self.zap.pscan.records_to_scan) > 0:
            print(f"Осталось записей для сканирования: {self.zap.pscan.records_to_scan}")
            time.sleep(2)

        # Получение результатов
        alerts = self.zap.core.alerts(baseurl=target_url)
        return alerts

    def active_scan(self, target_url):
        """Активное сканирование безопасности"""
        print(f"Запуск активного сканирования для {target_url}")

        # Запуск активного сканирования
        scan_id = self.zap.ascan.scan(target_url)

        # Мониторинг прогресса
        while int(self.zap.ascan.status(scan_id)) < 100:
            print(f"Прогресс сканирования: {self.zap.ascan.status(scan_id)}%")
            time.sleep(5)

        print("Активное сканирование завершено")

        # Получение результатов
```

```
        alerts = self.zap.core.alerts(baseurl=target_url)
        return alerts

    def generate_security_report(self, alerts, output_file='security_report.html'):
        """Генерация отчета по безопасности"""
        report_data = {
            'total_alerts': len(alerts),
            'high_risk': len([a for a in alerts if a['risk'] == 'High']),
            'medium_risk': len([a for a in alerts if a['risk'] == 'Medium']),
            'low_risk': len([a for a in alerts if a['risk'] == 'Low']),
            'alerts_by_plugin': {}
        }

        for alert in alerts:
            plugin = alert['pluginId']
            if plugin not in report_data['alerts_by_plugin']:
                report_data['alerts_by_plugin'][plugin] = 0
            report_data['alerts_by_plugin'][plugin] += 1

        # Генерация HTML отчета
        html_report = f"""
        <html>
        <head>
            <title>Security Scan Report</title>
            <style>
                body {{ font-family: Arial, sans-serif; margin: 20px; }}
                .high {{ color: red; font-weight: bold; }}
                .medium {{ color: orange; }}
                .low {{ color: green; }}
                table {{ border-collapse: collapse; width: 100%; }}
                th, td {{ border: 1px solid #ddd; padding: 8px; text-align: left; }}
                th {{ background-color: #f2f2f2; }}
            </style>
        </head>
        <body>
            <h1>Отчет сканирования безопасности</h1>
            <h2>Статистика</h2>
            <table>
                <tr><th>Метрика</th><th>Значение</th></tr>
                <tr><td>Всего предупреждений</td><td>{report_data['total_alerts']}</td></tr>
                <tr><td>Высокий риск</td><td class="high">{report_data['high_risk']}</td></tr>
                <tr><td>Средний риск</td><td class="medium">{report_data['medium_risk']}</td></tr>
                <tr><td>Низкий риск</td><td class="low">{report_data['low_risk']}</td></tr>
            </table>

            <h2>Детальные предупреждения</h2>
            <table>
                <tr><th>Риск</th><th>Название</th><th>Описание</th><th>URL</th></tr>
        """

        for alert in alerts:
            risk_class = alert['risk'].lower()
            html_report += f"""
                <tr>
                    <td class="{risk_class}">{alert['risk']}</td>
                    <td>{alert['name']}</td>
                    <td>{alert['description']}</td>
                    <td>{alert['url']}</td>
                </tr>
            """

        html_report += """
            </table>
        </body>
        </html>
        """

        with open(output_file, 'w', encoding='utf-8') as f:
            f.write(html_report)

        print(f"Отчет сохранен в: {output_file}")
        return report_data

# Использование
security_tester = SecurityTester()

target_url = "http://localhost:8080"
```

```
print("=== Запуск проверки безопасности ===")
passive_alerts = security_tester.passive_scan(target_url)
active_alerts = security_tester.active_scan(target_url)

all_alerts = passive_alerts + active_alerts
report = security_tester.generate_security_report(all_alerts)

print(f"Обнаружено предупреждений: {report['total_alerts']}")
print(f"Из них высокого риска: {report['high_risk']}")
```

Итоговое задание для самостоятельного выполнения:
Сценарий: Комплексное исследование веб-приложения на тестовом стенде.
Задание:
1. Подготовка тестовой среды:
python
```
# Развертывание полного тестового стенда
test_environment = {
    "web_application": "Демо-приложение для тестирования",
    "database": "MySQL с тестовыми данными",
    "monitoring": "Grafana + Prometheus",
    "load_balancer": "nginx",
    "security_tools": "OWASP ZAP, nmap"
}
```
2. План тестирования:
markdown
```
### Фаза 1: Функциональное тестирование
- [ ] Тестирование пользовательского интерфейса
- [ ] Тестирование API endpoints
- [ ] Тестирование бизнес-логики
- [ ] Тестирование обработки ошибок

### Фаза 2: Нагрузочное тестирование
- [ ] Тестирование производительности
- [ ] Стресс-тестирование
- [ ] Тестирование стабильности
- [ ] Тестирование масштабируемости

### Фаза 3: Тестирование безопасности
- [ ] Сканирование уязвимостей
- [ ] Тестирование аутентификации
- [ ] Тестирование авторизации
- [ ] Тестирование ввода данных
```
3. Автоматизация тестов:
python
```
def run_comprehensive_test_suite():
    """Запуск комплексного набора тестов"""

    test_results = {}

    # Функциональное тестирование
    functional_tester = FunctionalTests()
    test_results['functional'] = functional_tester.run_all_tests()

    # Нагрузочное тестирование
    performance_tester = PerformanceTester()
    test_results['performance'] = performance_tester.run_comprehensive_load_test()

    # Тестирование безопасности
    security_tester = SecurityTester()
    test_results['security'] = security_tester.run_full_security_scan()

    # Анализ и отчетность
    report_generator = TestReportGenerator(test_results)
    comprehensive_report = report_generator.generate_final_report()

    return comprehensive_report
```
4. Структура проекта:
text
```
software_testing_lab/
├── 01_test_environment/
│   ├── docker-compose.yml
│   ├── configuration/
│   └── monitoring/
├── 02_functional_tests/
│   ├── selenium_tests/
```

```
│   ├── api_tests/
│   └── unit_tests/
├── 03_performance_tests/
│   ├── jmeter_plans/
│   ├── locust_scripts/
│   └── results_analysis/
├── 04_security_tests/
│   ├── zap_scans/
│   ├── vulnerability_reports/
│   └── penetration_tests/
├── 05_test_reports/
│   ├── executive_summary/
│   ├── detailed_reports/
│   └── recommendations/
└── 06_automation_scripts/
    ├── deployment/
    ├── test_runners/
    └── monitoring/
```

5. Требования к результату:

Качество тестирования:

- Покрытие всех критических функциональностей
- Реалистичные сценарии нагрузочного тестирования
- Комплексная проверка безопасности

Документация:

- Детальные отчеты по каждому виду тестирования
- Рекомендации по устранению найденных проблем
- Метрики качества программного обеспечения

Автоматизация:

- Скрипты развертывания тестовой среды
- Автоматические тестовые сценарии
- Система мониторинга и сбора метрик

6. Критерии оценки:

Компонент	Критерии
Организация тестового стенда	Полнота, изолированность,
Качество функционального тестирования	Покрытие, автоматизация,
Нагрузочное тестирование	Реалистичность нагрузки, а
Тестирование безопасности	Глубина проверки, актуаль
Документация и отчетность	Полнота, ясность, практиче

Это занятие даст студентам комплексные практические навыки тестирования программного обеспечения, которые являются критически важными для QA-инженеров, DevOps-специалистов и разработчиков.
Практическое занятие 71 - 74. Разработка плана документа и согласование его с экспертами
Отлично! Разработаем комплексный план практического занятия по разработке и согласованию планов документов.

Практическое занятие 71-74: Разработка плана документа и согласование его с экспертами

Цель занятия: Освоить методику разработки структурированных планов технических документов и организовать процесс их согласования с subject matter experts (SME).

Программное обеспечение:

- **Планирование:** Microsoft Word, Google Docs, Notion, Confluence
- **Диаграммы:** Draw.io, Lucidchart, Miro
- **Согласование:** ReviewStudio, Filestage, PDF Annotator
- **Управление проектом:** Trello, Asana, Jira
- **Коммуникация:** Slack, Microsoft Teams, Calendly

Необходимые знания: Базовые навыки технического письма, понимание жизненного цикла документации.

План занятия и Ход работы

Часть 1: Основы планирования документа (≈4 часа)

Задача: Освоить принципы создания эффективных планов технических документов.

1. Компоненты плана документа:

python
```
class DocumentPlan:
    """Класс для создания плана технического документа"""
```

```
    def __init__(self, doc_title, doc_type, target_audience):
        self.title = doc_title
        self.type = doc_type  # manual, spec, report, etc.
        self.audience = target_audience
        self.structure = []
        self.metadata = {}

    def define_scope(self, scope_description, inclusions, exclusions):
        """Определение границ документа"""
        self.metadata['scope'] = {
            'description': scope_description,
            'in_scope': inclusions,
            'out_of_scope': exclusions
        }

    def create_structure(self, sections):
        """Создание иерархической структуры"""
        for section in sections:
            self.structure.append({
                'level': section['level'],
                'title': section['title'],
                'purpose': section.get('purpose', ''),
                'estimated_length': section.get('length', ''),
                'responsible': section.get('responsible', ''),
                'status': 'draft'
            })

    def generate_outline(self):
        """Генерация оглавления"""
        outline = f"# План документа: {self.title}\n\n"
        outline += f"**Тип:** {self.type}\n"
        outline += f"**Целевая аудитория:** {self.audience}\n\n"

        outline += "## Структура документа:\n"
        for section in self.structure:
            indent = "  " * (section['level'] - 1)
            outline += f"{indent}- {section['title']}\n"
            if section['purpose']:
                outline += f"{indent}  *Цель: {section['purpose']}*\n"

        return outline

# Пример использования
user_manual_plan = DocumentPlan(
    "Руководство пользователя CRM-системы",
    "user_manual",
    "менеджеры по продажам, администраторы"
)

user_manual_plan.define_scope(
    "Полное руководство по работе с CRM-системой",
    ["базовые операции", "отчетность", "администрирование"],
    ["техническая поддержка", "API документация"]
)

sections = [
    {'level': 1, 'title': 'Введение', 'purpose': 'Обзор системы'},
    {'level': 1, 'title': 'Начало работы', 'purpose': 'Быстрый старт'},
    {'level': 2, 'title': 'Регистрация', 'purpose': 'Создание учетной записи'},
    {'level': 2, 'title': 'Первоначальная настройка'},
    {'level': 1, 'title': 'Работа с клиентами', 'purpose': 'Управление базой клиентов'}
]

user_manual_plan.create_structure(sections)
print(user_manual_plan.generate_outline())
```

2. Практическое задание "Анализ требований":

```
markdown
### Задание: Разработка плана технического задания

**Продукт:** Мобильное приложение для доставки еды

**Требования:**
1. **Функциональные:**
   - Регистрация пользователей
   - Выбор ресторанов и блюд
   - Система оплаты
   - Отслеживание заказов
```

2. **Нефункциональные:**
   - Производительность: < 3 сек на операцию
   - Безопасность: шифрование данных
   - Совместимость: iOS 13+, Android 8+

3. **Аудитория:**
   - Конечные пользователи (документация UI)
   - Разработчики (API документация)
   - Администраторы (руководство по администрированию)

Результат: Детальный план ТЗ с разбивкой по разделам

Часть 2: Методологии структурирования документов (≈4 часа)
Задача: Изучить различные подходы к организации технической документации.
1. Методология Diátaxis:
python
```
class DiataxisFramework:
    """Реализация методологии Diátaxis для технической документации"""

    def __init__(self):
        self.categories = {
            'tutorials': {
                'purpose': 'Обучение через выполнение',
                'structure': ['цель', 'предварительные требования', 'шаги', 'резюме'],
                'examples': ['быстрый старт', 'пошаговые руководства']
            },
            'how-to_guides': {
                'purpose': 'Решение конкретных задач',
                'structure': ['проблема', 'решение', 'пример'],
                'examples': ['решение проблем', 'рецепты']
            },
            'reference': {
                'purpose': 'Техническая информация',
                'structure': ['описание', 'параметры', 'примеры', 'примечания'],
                'examples': ['API документация', 'справочники']
            },
            'explanation': {
                'purpose': 'Понимание концепций',
                'structure': ['введение', 'концепция', 'детали', 'выводы'],
                'examples': ['архитектура', 'объяснительные статьи']
            }
        }

    def create_document_structure(self, doc_type, topic):
        """Создание структуры документа по методологии Diátaxis"""
        if doc_type not in self.categories:
            raise ValueError(f"Неизвестный тип документа: {doc_type}")

        category = self.categories[doc_type]
        structure = {
            'type': doc_type,
            'purpose': category['purpose'],
            'sections': category['structure'],
            'topic': topic,
            'best_practices': self.get_best_practices(doc_type)
        }

        return structure

    def get_best_practices(self, doc_type):
        """Рекомендации по созданию документа"""
        practices = {
            'tutorials': [
                'Фокусируйтесь на результате',
                'Используйте последовательные шаги',
                'Включайте скриншоты и примеры',
                'Проверяйте каждую инструкцию'
            ],
            'how-to_guides': [
                'Начинайте с проблемы',
                'Давайте конкретное решение',
                'Включайте примеры кода',
                'Учитывайте альтернативные сценарии'
            ],
            'reference': [
```

```
            'Будьте точными и полными',
            'Используйте стандартный формат',
            'Включайте примеры использования',
            'Обновляйте при изменениях API'
        ],
        'explanation': [
            'Объясняйте "почему", а не "как"',
            'Используйте аналогии',
            'Связывайте с контекстом',
            'Избегайте инструкций'
        ]
    }
    return practices.get(doc_type, [])

# Использование методологии
diataxis = DiataxisFramework()

# Создание структуры руководства
tutorial_structure = diataxis.create_document_structure(
    'tutorials',
    'Создание первого отчета в системе'
)

print("Структура руководства:")
for section in tutorial_structure['sections']:
    print(f"- {section}")

print("\nРекомендации:")
for practice in tutorial_structure['best_practices']:
    print(f"• {practice}")
```

2. Практическое задание "Применение методологий":

```
python
def create_comprehensive_doc_plan(product_name, doc_types):
    """Создание комплексного плана документации для продукта"""

    doc_plan = {
        'product': product_name,
        'documentation_set': [],
        'timeline': {},
        'stakeholders': []
    }

    diataxis = DiataxisFramework()

    for doc_type in doc_types:
        structure = diataxis.create_document_structure(doc_type['type'], doc_type['topic'])
        doc_plan['documentation_set'].append({
            'document_type': doc_type['type'],
            'topic': doc_type['topic'],
            'target_audience': doc_type.get('audience', 'пользователи'),
            'estimated_pages': doc_type.get('pages', 10),
            'structure': structure,
            'status': 'planned'
        })

    return doc_plan

# Пример использования
product_docs = create_comprehensive_doc_plan("Аналитическая платформа DataInsight", [
    {'type': 'tutorials', 'topic': 'Первая настройка системы', 'audience': 'новые пользователи', 'pages': 15},
    {'type': 'how-to_guides', 'topic': 'Создание пользовательских отчетов', 'audience': 'аналитики', 'pages': 25},
    {'type': 'reference', 'topic': 'API документация', 'audience': 'разработчики', 'pages': 50},
    {'type': 'explanation', 'topic': 'Архитектура системы', 'audience': 'технические руководители', 'pages': 20}
])

print("Комплексный план документации:")
for doc in product_docs['documentation_set']:
    print(f"\n{doc['document_type'].upper()}: {doc['topic']}")
    print(f"Аудитория: {doc['target_audience']}")
    print(f"Статус: {doc['status']}")
```

Часть 3: Подготовка к согласованию (≈4 часа)

Задача: Научиться готовить материалы для эффективного согласования с экспертами.

1. Шаблон плана для согласования:

```
python
class ReviewPackage:
```

```
"""Пакет материалов для согласования плана документа"""

def __init__(self, doc_plan, stakeholders):
    self.doc_plan = doc_plan
    self.stakeholders = stakeholders
    self.review_materials = {}

def create_review_package(self):
    """Создание полного пакета для согласования"""
    self.review_materials = {
        'executive_summary': self.create_executive_summary(),
        'detailed_plan': self.create_detailed_plan(),
        'review_questions': self.prepare_review_questions(),
        'timeline': self.create_review_timeline(),
        'feedback_template': self.create_feedback_template()
    }
    return self.review_materials

def create_executive_summary(self):
    """Создание краткого обзора для руководителей"""
    summary = f"""
    КРАТКИЙ ОБЗОР ПЛАНА ДОКУМЕНТАЦИИ

    Продукт: {self.doc_plan['product']}

    Объем документации:
    - Всего документов: {len(self.doc_plan['documentation_set'])}
    - Общий объем: {sum(doc['estimated_pages'] for doc in self.doc_plan['documentation_set'])} страниц
    - Целевые аудитории: {', '.join(set(doc['target_audience'] for doc in self.doc_plan['documentation_set']))}

    Ключевые разделы:
    {self.get_key_sections()}

    Сроки разработки: 4-6 недель
    """
    return summary

def prepare_review_questions(self):
    """Подготовка вопросов для экспертов"""
    questions = {
        'content_questions': [
            "Полностью ли покрыты потребности вашей аудитории?",
            "Какие разделы требуют более детального раскрытия?",
            "Есть ли избыточная информация, которую можно исключить?",
            "Соответствует ли структура вашим рабочим процессам?"
        ],
        'technical_questions': [
            "Достаточна ли техническая глубина описаний?",
            "Верно ли определены технические предпосылки?",
            "Соответствует ли документация текущей версии продукта?",
            "Какие API/функции требуют более детального описания?"
        ],
        'usability_questions': [
            "Удобна ли предложенная структура для поиска информации?",
            "Достаточно ли примеров и практических сценариев?",
            "Соответствует ли уровень сложности целевой аудитории?",
            "Какие улучшения навигации вы можете предложить?"
        ]
    }
    return questions

def create_feedback_template(self):
    """Создание шаблона для сбора обратной связи"""
    template = """
    ШАБЛОН ОБРАТНОЙ СВЯЗИ ПО ПЛАНУ ДОКУМЕНТАЦИИ

    Эксперт: [Имя и должность]
    Дата: [Дата заполнения]

    1. ОБЩАЯ ОЦЕНКА СТРУКТУРЫ:
    - Сильные стороны:
    - Области для улучшения:

    2. КОНКРЕТНЫЕ ЗАМЕЧАНИЯ ПО РАЗДЕЛАМ:

    Раздел: [Название раздела]
    - Предлагаемые изменения:
```

```
        - Дополнительные темы для включения:
        - Рекомендации по глубине раскрытия:

        3. ТЕХНИЧЕСКАЯ ТОЧНОСТЬ:
        - Выявленные неточности:
        - Отсутствующие технические детали:
        - Рекомендации по терминологии:

        4. ПРАКТИЧЕСКАЯ ЦЕННОСТЬ:
        - Насколько полезен документ для вашей работы:
        - Предложения по улучшению usability:
        - Рекомендации по примерам и кейсам:

        5. ДОПОЛНИТЕЛЬНЫЕ КОММЕНТАРИИ:
        """
        return template

# Создание пакета для согласования
review_package = ReviewPackage(product_docs, [
    'технический директор',
    'ведущий разработчик',
    'менеджер продукта',
    'представитель пользователей'
])

materials = review_package.create_review_package()
print("Пакет для согласования создан:")
for material in materials.keys():
    print(f"- {material}")
```

2. Практическое задание "Подготовка встречи":

```
python
def schedule_expert_review(plan_package, experts_availability):
    """Планирование встреч по согласованию"""

    meeting_plan = {
        'kickoff_meeting': {
            'purpose': 'Общее представление плана',
            'duration': '60 минут',
            'participants': 'все стейкхолдеры',
            'agenda': [
                'Представление общего подхода',
                'Обзор структуры документации',
                'Определение ролей и ответственности',
                'Согласование сроков'
            ]
        },
        'technical_review': {
            'purpose': 'Детальный технический разбор',
            'duration': '90 минут',
            'participants': 'технические эксперты',
            'agenda': [
                'Обзор технических разделов',
                'Обсуждение API документации',
                'Проверка терминологии',
                'Согласование технической глубины'
            ]
        },
        'usability_review': {
            'purpose': 'Оценка удобства использования',
            'duration': '60 минут',
            'participants': 'представители пользователей',
            'agenda': [
                'Оценка структуры с точки зрения пользователя',
                'Обсуждение примеров и сценариев',
                'Рекомендации по улучшению навигации'
            ]
        }
    }

    # Создание приглашений
    invitations = []
    for meeting_type, details in meeting_plan.items():
        invitation = f"""
        ПРИГЛАШЕНИЕ НА ВСТРЕЧУ: {details['purpose']}

        Уважаемые эксперты,
```

Приглашаем вас принять участие в согласовании плана документации для продукта "{plan_package.doc_plan['product']}".

Дата и время: [уточняется согласно вашему расписанию]
Продолжительность: {details['duration']}
Участники: {details['participants']}

Повестка дня:
{'\n'.join(f'- {item}' for item in details['agenda'])}

Подготовительные материалы будут направлены за 2 рабочих дня до встречи.

С уважением,
Команда технической документации
"""
invitations.append(invitation)

return meeting_plan, invitations

Часть 4: Проведение согласования (≈4 часа)
Задача: Освоить техники проведения встреч по согласованию и обработки обратной связи.
1. Инструменты для сбора обратной связи:

```python
class FeedbackManager:
    """Управление обратной связью от экспертов"""

    def __init__(self):
        self.feedback_data = []
        self.action_items = []

    def collect_feedback(self, expert_name, role, feedback):
        """Сбор структурированной обратной связи"""
        feedback_record = {
            'expert': expert_name,
            'role': role,
            'date': '2024-01-15',
            'feedback': feedback,
            'priority': 'medium',  # low, medium, high, critical
            'status': 'new'
        }
        self.feedback_data.append(feedback_record)

    def analyze_feedback_patterns(self):
        """Анализ паттернов в обратной связи"""
        themes = {}

        for record in self.feedback_data:
            # Простой анализ ключевых тем
            feedback_text = record['feedback'].lower()

            if 'структур' in feedback_text:
                themes.setdefault('structure', []).append(record)
            if 'техни' in feedback_text:
                themes.setdefault('technical', []).append(record)
            if 'пример' in feedback_text or 'кейс' in feedback_text:
                themes.setdefault('examples', []).append(record)
            if 'навигац' in feedback_text or 'поиск' in feedback_text:
                themes.setdefault('navigation', []).append(record)

        return themes

    def create_action_plan(self):
        """Создание плана действий по обратной связи"""
        themes = self.analyze_feedback_patterns()

        for theme, records in themes.items():
            action_item = {
                'theme': theme,
                'description': self.get_theme_description(theme),
                'actions': [],
                'owner': 'technical_writer',
                'deadline': '2024-01-25',
                'status': 'pending'
            }

            for record in records:
```

```
                if record['priority'] in ['high', 'critical']:
                    action_item['actions'].append(
                        f"Учесть комментарий от {record['expert']}: {record['feedback'][:100]}..."
                    )

            if action_item['actions']:
                self.action_items.append(action_item)

        return self.action_items

    def get_theme_description(self, theme):
        """Описание темы для плана действий"""
        descriptions = {
            'structure': 'Изменения в структуре документа',
            'technical': 'Технические правки и уточнения',
            'examples': 'Добавление примеров и кейсов',
            'navigation': 'Улучшение навигации и поиска'
        }
        return descriptions.get(theme, 'Общие улучшения')

    def generate_feedback_report(self):
        """Генерация отчета по обратной связи"""
        report = f"""
        ОТЧЕТ ПО РЕЗУЛЬТАТАМ СОГЛАСОВАНИЯ

        Всего получено отзывов: {len(self.feedback_data)}
        Участвовало экспертов: {len(set(r['expert'] for r in self.feedback_data))}

        РАСПРЕДЕЛЕНИЕ ПО ТЕМАМ:
        """

        themes = self.analyze_feedback_patterns()
        for theme, records in themes.items():
            report += f"\n- {theme}: {len(records)} отзывов"

        report += "\n\nПЛАН ДЕЙСТВИЙ:"
        for action in self.action_items:
            report += f"\n\n{action['theme'].upper()}:"
            for act in action['actions']:
                report += f"\n• {act}"

        return report

# Пример использования
feedback_manager = FeedbackManager()

# Сбор фидбека от экспертов
feedback_manager.collect_feedback(
    "Анна Петрова",
    "Technical Lead",
    "Необходимо добавить раздел по устранению неполадок API. Также стоит расширить примеры использования."
)

feedback_manager.collect_feedback(
    "Иван Сидоров",
    "Product Manager",
    "Структура хорошая, но нужно добавить больше бизнес-кейсов. Раздел 'Начало работы' слишком технический."
)

# Анализ и создание плана действий
action_plan = feedback_manager.create_action_plan()
report = feedback_manager.generate_feedback_report()

print(report)
```

2. Практическое задание "Модерация встречи":

python

```
class MeetingFacilitator:
    """Класс для модерации встреч по согласованию"""

    def __init__(self, meeting_type, participants):
        self.meeting_type = meeting_type
        self.participants = participants
        self.notes = []
        self.decisions = []

    def facilitate_meeting(self, agenda):
        """Проведение встречи с фиксацией результатов"""
```

```
        meeting_minutes = {
            'meeting_type': self.meeting_type,
            'date': '2024-01-15',
            'participants': self.participants,
            'agenda_items': [],
            'decisions_made': [],
            'action_items': []
        }

        for agenda_item in agenda:
            discussion_summary = self.facilitate_discussion(agenda_item)
            meeting_minutes['agenda_items'].append({
                'topic': agenda_item,
                'discussion': discussion_summary,
                'decisions': self.get_decisions(),
                'actions': self.get_action_items()
            })

        return meeting_minutes

    def facilitate_discussion(self, topic):
        """Модерация обсуждения конкретной темы"""
        facilitation_techniques = [
            "Начните с открытого вопроса по теме",
            "Выслушайте всех участников",
            "Резюмируйте ключевые точки",
            "Определите зоны согласия и разногласия",
            "Сформулируйте конкретные предложения"
        ]

        # Имитация обсуждения
        discussion = f"Обсуждение темы '{topic}':\n"
        for technique in facilitation_techniques:
            discussion += f"- {technique}\n"

        return discussion

    def get_decisions(self):
        """Фиксация принятых решений"""
        decisions = [
            "Структура документа утверждена с изменениями",
            "Добавлен новый раздел по устранению неполадок",
            "Уточнена терминология в технических разделах"
        ]
        return decisions

    def get_action_items(self):
        """Формирование задач по результатам обсуждения"""
        actions = [
            {"task": "Добавить раздел Troubleshooting", "owner": "technical_writer", "deadline": "2024-01-25"},
            {"task": "Предоставить дополнительные бизнес-кейсы", "owner": "product_manager", "deadline": "2024-01-20"},
            {"task": "Review технических терминов", "owner": "tech_lead", "deadline": "2024-01-22"}
        ]
        return actions

# Проведение встречи
facilitator = MeetingFacilitator(
    "Согласование плана технической документации",
    ["Технический писатель", "Tech Lead", "Product Manager", "UX Designer"]
)

agenda = [
    "Обзор предложенной структуры",
    "Обсуждение технических разделов",
    "Оценка usability и навигации",
    "Согласование сроков и ответственности"
]

meeting_results = facilitator.facilitate_meeting(agenda)

print("Результаты встречи:")
print(f"Тип встречи: {meeting_results['meeting_type']}")
print(f"Участники: {', '.join(meeting_results['participants'])}")
print(f"\nПринятые решения: {len(meeting_results['decisions_made'])}")
print(f"Запланированные действия: {len(meeting_results['action_items'])}")
```

Итоговое задание для самостоятельного выполнения:
Сценарий: Разработка и согласование комплексного плана документации для нового SaaS-продукта.
Задание:
1. Разработка плана документации:
python
```
# Создание комплексного плана
saas_product_plan = DocumentPlan(
    "Документация платформы аналитики CloudAnalytics",
    "comprehensive_documentation_set",
    "бизнес-пользователи, аналитики, администраторы, разработчики"
)

# Определение структуры по методологии Diátaxis
documentation_structure = {
    'getting_started': {
        'type': 'tutorials',
        'sections': ['быстрый старт', 'первая настройка', 'создание первого отчета'],
        'audience': 'новые пользователи'
    },
    'user_guides': {
        'type': 'how-to_guides',
        'sections': ['работа с отчетами', 'настройка дашбордов', 'управление данными'],
        'audience': 'регулярные пользователи'
    },
    'admin_manual': {
        'type': 'reference',
        'sections': ['управление пользователями', 'настройки системы', 'безопасность'],
        'audience': 'администраторы'
    },
    'api_reference': {
        'type': 'reference',
        'sections': ['аутентификация', 'endpoints', 'примеры запросов'],
        'audience': 'разработчики'
    }
}
```
2. Подготовка пакета согласования:
python
```
# Создание полного пакета для согласования
review_package = ReviewPackage(saas_product_plan, [
    {'name': 'CTO', 'role': 'технический руководитель'},
    {'name': 'Head of Product', 'role': 'руководитель продукта'},
    {'name': 'Lead Developer', 'role': 'ведущий разработчик'},
    {'name': 'Customer Success Manager', 'role': 'представитель пользователей'}
])

materials = review_package.create_review_package()
```
3. Организация процесса согласования:
markdown
```
### План согласования:

**Неделя 1: Подготовка**
- [ ] Разработка детального плана документации
- [ ] Подготовка пакета материалов для согласования
- [ ] Назначение встреч с экспертами

**Неделя 2: Проведение согласования**
- [ ] Kick-off встреча (все стейкхолдеры)
- [ ] Технический review (разработчики)
- [ ] Бизнес review (продакт-менеджеры)
- [ ] Пользовательский review (Customer Success)

**Неделя 3: Обработка обратной связи**
- [ ] Анализ и систематизация замечаний
- [ ] Внесение изменений в план
- [ ] Финальное утверждение
```
4. Структура проекта:
text
```
documentation_plan_project/
├── 01_initial_plan/
│   ├── product_analysis.md
│   ├── audience_analysis.md
│   ├── documentation_plan_draft.md
│   └── diataxis_structure.md
├── 02_review_materials/
│   ├── executive_summary.pdf
│   ├── detailed_plan.pdf
│   ├── review_questions.md
```

```
│   └── feedback_templates/
├── 03_meeting_materials/
│   ├── kickoff_presentation.pptx
│   ├── technical_review_agenda.md
│   ├── business_review_notes.md
│   └── user_review_feedback.md
├── 04_feedback_analysis/
│   ├── feedback_collection.csv
│   ├── theme_analysis.md
│   ├── action_plan.md
│   └── revised_plan.md
└── 05_final_deliverables/
    ├── approved_plan.pdf
    ├── implementation_roadmap.md
    ├── stakeholder_signoff.md
    └── project_retrospective.md
```

5. Требования к результату:

Качество плана:

- Полное покрытие потребностей всех целевых аудиторий
- Соответствие методологиям технического письма
- Учет технических и бизнес-требований

Процесс согласования:

- Вовлечение всех ключевых стейкхолдеров
- Эффективный сбор и обработка обратной связи
- Документирование принятых решений

Документация процесса:

- Подробные отчеты по каждой встрече
- Систематизированная обратная связь
- Четкий план действий по доработке

6. Критерии оценки:

Компонент	Критерии
Качество плана документации	Полнота, структура, соответствие м
Организация процесса согласования	Подготовка материалов, планирован
Эффективность согласования	Вовлечение экспертов, сбор обратно
Обработка результатов	Анализ фидбека, план доработок
Документирование	Полнота отчетов, профессиональное

Это занятие даст студентам практические навыки разработки профессиональных планов документации и организации процессов их согласования, что является ключевой компетенцией для технических писателей и документалистов.
Практическое занятие 75 - 78. Составление информационных текстовых файлов
Отлично! Разработаем комплексный план практического занятия по созданию информационных текстовых файлов.

Практическое занятие 75-78: Составление информационных текстовых файлов
Цель занятия: Освоить принципы создания структурированных информационных текстовых файлов различных форматов для технической документации, инструкций и справочных материалов.
Программное обеспечение:

- **Текстовые редакторы:** VS Code, Sublime Text, Notepad++
- **Форматы разметки:** Markdown, reStructuredText, AsciiDoc
- **Системы контроля версий:** Git, GitHub, GitLab
- **Инструменты валидации:** textlint, markdownlint, vale
- **Сборка документации:** MkDocs, Sphinx, Jekyll

Необходимые знания: Базовые навыки работы с текстовыми редакторами, понимание структуры документов.

План занятия и Ход работы
Часть 1: Основы работы с текстовыми форматами (≈4 часа)
Задача: Освоить основные текстовые форматы и их применение.
1. Сравнение текстовых форматов:

Формат	Преимущества	Недостатки	Лучшее применение
Plain Text	Универсальность, простота	Нет форматирования	Конфиги, логи, заметки
Markdown	Простота, популярность	Ограниченные возможности	Документация, README
reStructuredText	Мощные возможности	Сложный синтаксис	Техническая документация
AsciiDoc	Богатые функции	Меньше инструментов	Книги, большие проекты

2. Базовые шаблоны текстовых файлов:

```
python
class TextFileTemplates:
    """Шаблоны для различных типов текстовых файлов"""

    @staticmethod
    def create_readme(project_name, description, installation, usage):
        """Шаблон README файла"""
        template = f"""# {project_name}

{description}

## Оглавление
- [Установка](#установка)
- [Использование](#использование)
- [Разработка](#разработка)
- [Лицензия](#лицензия)

## Установка

{installation}

## Использование

{usage}

## Разработка

### Требования
- Python 3.8+
- Зависимости из requirements.txt

### Установка для разработки
```bash
git clone https://github.com/username/{project_name.lower()}
cd {project_name.lower()}
pip install -r requirements.txt
```

Лицензия

Этот проект лицензирован under the MIT License - see LICENSE file for details.
"""
        return template

    @staticmethod
    def create_changelog():
        """Шаблон файла изменений"""
        template = """# История изменений

Все значимые изменения в этом проекте будут документированы в этом файле.

Формат основан на Keep a Changelog,
и этот проект придерживается Semantic Versioning.

[Невыпущено]

Добавлено

-

Изменено

-

Исправлено

-

[1.0.0] - 2024-01-15
```

Добавлено
- Первоначальный релиз
- Базовая функциональность
- Документация
"""
return template
text
@staticmethod
def create_config_file(config_type="yaml"):
"""Шаблон конфигурационного файла"""
if config_type == "yaml":
template = """# Конфигурация приложения
app:
name: "Мое приложение"
version: "1.0.0"
debug: false
database:
host: "localhost"
port: 5432
name: "myapp_db"
username: "admin"
password: "secret"
logging:
level: "INFO"
file: "app.log"
max_size: "10MB"
"""
elif config_type == "ini":
template = """; Конфигурация приложения
[app]
name = Мое приложение
version = 1.0.0
debug = false
[database]
host = localhost
port = 5432
name = myapp_db
username = admin
password = secret
[logging]
level = INFO
file = app.log
max_size = 10MB
"""
return template
Пример использования
readme_content = TextFileTemplates.create_readme(
"MyAwesomeProject",
"Это потрясающий проект для демонстрации возможностей.",
"bash\npip install myawesomeproject\n",
"python\nimport myawesomeproject\nresult = myawesomeproject.do_something()\n"
)
print("Создан README файл:")
print(readme_content[:200] + "...")
text

3. Практическое задание "Создание базовых файлов":
```markdown
### Задание: Создание набора текстовых файлов для проекта

**Проект:** "Анализатор логов"

**Требуемые файлы:**
1. **README.md** - основная документация
2. **CHANGELOG.md** - история изменений
3. **config.yaml** - конфигурация приложения
4. **CONTRIBUTING.md** - руководство для контрибьюторов
5. **TODO.md** - список планируемых улучшений

**Критерии качества:**
- Единый стиль оформления
- Полнота информации
- Соответствие стандартам
- Читаемость и структурированность

---

**Часть 2: Работа с Markdown (≈4 часа)**
```

Задача: Освоить расширенные возможности Markdown для технической документации.
1. Расширенный Markdown с Mermaid:
markdown
Техническая документация API

Обзор системы

```mermaid
graph TD
    A[Клиент] --> B[API Gateway]
    B --> C[Сервис аутентификации]
    B --> D[Сервис пользователей]
    B --> E[Сервис заказов]
    D --> F[(База пользователей)]
    E --> G[(База заказов)]
```

API Endpoints
Получение информации о пользователе
HTTP Запрос:
http
GET /api/v1/users/{user_id}
Authorization: Bearer {token}
Параметры:

Парамет р	Тип	Обязательный	Описание
user_id	integer	Да	ID пользователя
fields	string	Нет	Поля для включения в ответ

Пример ответа:
json
```
{
    "status": "success",
    "data": {
        "id": 123,
        "username": "john_doe",
        "email": "john@example.com",
        "created_at": "2024-01-15T10:30:00Z"
    }
}
```

Таблица кодов ответов

Код	Описание	Действие
200	OK	Успешный
400	Bad Request	Неверные п
401	Unauthorized	Требуется а
404	Not Found	Ресурс не н
500	Internal Error	Ошибка сер

Безопасность
Важно: Все API запросы должны использовать HTTPS.
Токены аутентификации должны храниться безопасно.
Пример использования
python
```
import requests

def get_user_info(user_id, token):
    headers = {'Authorization': f'Bearer {token}'}
    response = requests.get(
        f'https://api.example.com/v1/users/{user_id}',
        headers=headers
```

```
    )

    if response.status_code == 200:
        return response.json()['data']
    else:
        raise Exception(f"Ошибка API: {response.status_code}")
text

**2. Python скрипт для генерации Markdown документации:**
```python
import os
from datetime import datetime
from typing import List, Dict

class MarkdownGenerator:
    """Генератор Markdown документации"""

    def __init__(self, project_name: str, version: str = "1.0.0"):
        self.project_name = project_name
        self.version = version
        self.sections = []

    def add_section(self, title: str, content: str, level: int = 2):
        """Добавление раздела в документ"""
        self.sections.append({
            'title': title,
            'content': content,
            'level': level
        })

    def create_table(self, headers: List[str], rows: List[List[str]]) -> str:
        """Создание Markdown таблицы"""
        table = "| " + " | ".join(headers) + " |\n"
        table += "|" + "|".join([" --- " for _ in headers]) + "|\n"

        for row in rows:
            table += "| " + " | ".join(row) + " |\n"

        return table

    def generate_document(self) -> str:
        """Генерация полного документа"""
        doc = f"""# {self.project_name}

*Версия {self.version} - Последнее обновление: {datetime.now().strftime('%Y-%m-%d')}*

## Оглавление

"""
        # Автогенерация оглавления
        for i, section in enumerate(self.sections, 1):
            indent = "  " * (section['level'] - 2)
            doc += f"{indent}- [{section['title']}](#{section['title'].lower().replace(' ', '-')})\n"

        doc += "\n"

        # Добавление разделов
        for section in self.sections:
            hashes = "#" * section['level']
            doc += f"{hashes} {section['title']}\n\n"
            doc += section['content'] + "\n\n"

        return doc

    def save_to_file(self, filename: str):
        """Сохранение документа в файл"""
        content = self.generate_document()

        os.makedirs(os.path.dirname(filename), exist_ok=True)
        with open(filename, 'w', encoding='utf-8') as f:
            f.write(content)

        print(f"Документ сохранен: {filename}")

# Пример использования
doc_gen = MarkdownGenerator("Система управления задачами", "2.1.0")
```
```

```
# Добавление разделов
doc_gen.add_section("Введение",
"""Система управления задачами позволяет эффективно организовывать
и отслеживать выполнение проектов.""")

api_table_headers = ["Метод", "Endpoint", "Описание", "Аутентификация"]
api_table_rows = [
    ["GET", "/api/tasks", "Получить список задач", "Требуется"],
    ["POST", "/api/tasks", "Создать новую задачу", "Требуется"],
    ["PUT", "/api/tasks/{id}", "Обновить задачу", "Требуется"],
    ["DELETE", "/api/tasks/{id}", "Удалить задачу", "Требуется"]
]

api_content = "## API Endpoints\n\n" + doc_gen.create_table(api_table_headers, api_table_rows)
doc_gen.add_section("API Документация", api_content)

doc_gen.add_section("Установка",
"""### Требования
- Python 3.8+
- PostgreSQL 12+

### Установка
```bash
git clone https://github.com/example/task-manager
cd task-manager
pip install -r requirements.txt
```""")

# Сохранение документа
doc_gen.save_to_file("docs/user_manual.md")
```

3. Практическое задание "Документация API":

```
python
def create_api_documentation(api_endpoints):
    """Создание полной документации API"""

    doc = "# API Documentation\n\n"

    for endpoint in api_endpoints:
        doc += f"## {endpoint['method']} {endpoint['path']}\n\n"
        doc += f"{endpoint['description']}\n\n"

        if endpoint.get('parameters'):
            doc += "### Параметры\n\n"
            param_headers = ["Имя", "Тип", "Обязательный", "Описание"]
            param_rows = []

            for param in endpoint['parameters']:
                param_rows.append([
                    param['name'],
                    param['type'],
                    "Да" if param.get('required', False) else "Нет",
                    param.get('description', '')
                ])

            doc += create_markdown_table(param_headers, param_rows)

        if endpoint.get('example_request'):
            doc += "### Пример запроса\n\n"
            doc += f"```http\n{endpoint['example_request']}\n```\n\n"

        if endpoint.get('example_response'):
            doc += "### Пример ответа\n\n"
            doc += f"```json\n{endpoint['example_response']}\n```\n\n"

        doc += "---\n\n"

    return doc

# Пример данных API
api_data = [
    {
        'method': 'POST',
        'path': '/api/v1/tasks',
        'description': 'Создание новой задачи',
        'parameters': [
            {'name': 'title', 'type': 'string', 'required': True, 'description': 'Название задачи'},
            {'name': 'description', 'type': 'string', 'required': False, 'description': 'Описание задачи'},
```

```
            {'name': 'priority', 'type': 'integer', 'required': False, 'description': 'Приоритет (1-5)'}
        ],
        'example_request': 'POST /api/v1/tasks\nContent-Type: application/json\n\n{\n  "title": "Важная задача",\n  "priority": 1\n}',
        'example_response': '{\n  "id": 123,\n  "title": "Важная задача",\n  "status": "pending",\n  "created_at": "2024-01-15T10:00:00Z"\n}'
    }
]

api_doc = create_api_documentation(api_data)
print(api_doc)
```

Часть 3: Структурированные форматы данных (≈4 часа)
Задача: Освоить создание и работу с структурированными текстовыми форматами.
1. Работа с YAML и JSON:
python
```
import yaml
import json
from typing import Dict, Any

class StructuredDataManager:
    """Управление структурированными данными в различных форматах"""

    @staticmethod
    def create_project_config(project_name: str, config_data: Dict[str, Any]) -> str:
        """Создание конфигурационного файла проекта"""

        config_template = {
            'project': {
                'name': project_name,
                'version': config_data.get('version', '1.0.0'),
                'description': config_data.get('description', ''),
                'authors': config_data.get('authors', []),
            },
            'database': {
                'host': config_data.get('db_host', 'localhost'),
                'port': config_data.get('db_port', 5432),
                'name': config_data.get('db_name', f"{project_name.lower()}_db"),
            },
            'api': {
                'host': config_data.get('api_host', '0.0.0.0'),
                'port': config_data.get('api_port', 8000),
                'debug': config_data.get('debug', False),
            },
            'logging': {
                'level': config_data.get('log_level', 'INFO'),
                'file': config_data.get('log_file', 'app.log'),
                'format': config_data.get('log_format', '%(asctime)s - %(levelname)s - %(message)s'),
            }
        }

        return yaml.dump(config_template, default_flow_style=False, allow_unicode=True, indent=2)

    @staticmethod
    def create_openapi_spec(api_info: Dict[str, Any]) -> str:
        """Создание OpenAPI спецификации"""

        openapi_template = {
            'openapi': '3.0.0',
            'info': {
                'title': api_info.get('title', 'API Documentation'),
                'version': api_info.get('version', '1.0.0'),
                'description': api_info.get('description', ''),
                'contact': {
                    'name': api_info.get('contact_name', 'API Support'),
                    'email': api_info.get('contact_email', 'support@example.com')
                }
            },
            'servers': [
                {
                    'url': api_info.get('server_url', 'http://localhost:8000'),
                    'description': api_info.get('server_description', 'Development server')
                }
            ],
            'paths': api_info.get('paths', {}),
            'components': {
```

```
                'schemas': api_info.get('schemas', {}),
                'securitySchemes': api_info.get('security_schemes', {})
            }
        }

        return json.dumps(openapi_template, ensure_ascii=False, indent=2)

    @staticmethod
    def validate_yaml_file(file_path: str) -> bool:
        """Валидация YAML файла"""
        try:
            with open(file_path, 'r', encoding='utf-8') as f:
                yaml.safe_load(f)
            return True
        except yaml.YAMLError as e:
            print(f"Ошибка в YAML файле {file_path}: {e}")
            return False

    @staticmethod
    def convert_json_to_yaml(json_file_path: str, yaml_file_path: str):
        """Конвертация JSON в YAML"""
        try:
            with open(json_file_path, 'r', encoding='utf-8') as f:
                json_data = json.load(f)

            with open(yaml_file_path, 'w', encoding='utf-8') as f:
                yaml.dump(json_data, f, default_flow_style=False, allow_unicode=True, indent=2)

            print(f"Файл конвертирован: {json_file_path} -> {yaml_file_path}")

        except Exception as e:
            print(f"Ошибка конвертации: {e}")

# Пример использования
config_data = {
    'version': '2.1.0',
    'description': 'Система управления проектами',
    'authors': ['Анна Петрова', 'Иван Сидоров'],
    'db_host': 'db.example.com',
    'db_port': 5432,
    'db_name': 'project_management',
    'api_port': 8080,
    'debug': True,
    'log_level': 'DEBUG'
}

yaml_config = StructuredDataManager.create_project_config("ProjectManager", config_data)
print("YAML конфигурация:")
print(yaml_config)

# Создание OpenAPI спецификации
api_info = {
    'title': 'Project Management API',
    'version': '1.0.0',
    'description': 'API для управления проектами и задачами',
    'contact_name': 'Техническая поддержка',
    'contact_email': 'tech@example.com',
    'server_url': 'https://api.example.com/v1',
    'paths': {
        '/projects': {
            'get': {
                'summary': 'Получить список проектов',
                'responses': {
                    '200': {
                        'description': 'Успешный ответ',
                        'content': {
                            'application/json': {
                                'schema': {
                                    'type': 'array',
                                    'items': {
                                        '$ref': '#/components/schemas/Project'
                                    }
                                }
                            }
                        }
                    }
                }
            }
```

```
            }
        }
    }
}

openapi_spec = StructuredDataManager.create_openapi_spec(api_info)
print("\nOpenAPI спецификация:")
print(openapi_spec[:200] + "...")
```

2. Практическое задание "Конфигурационные файлы":

```
python
def create_docker_compose(services_config):
    """Создание docker-compose.yml файла"""

    compose_template = {
        'version': '3.8',
        'services': {},
        'networks': {
            'app-network': {
                'driver': 'bridge'
            }
        },
        'volumes': {}
    }

    for service_name, config in services_config.items():
        compose_template['services'][service_name] = {
            'image': config.get('image'),
            'container_name': config.get('container_name', service_name),
            'ports': config.get('ports', []),
            'environment': config.get('environment', {}),
            'volumes': config.get('volumes', []),
            'networks': ['app-network'],
            'depends_on': config.get('depends_on', [])
        }

        if config.get('build'):
            compose_template['services'][service_name]['build'] = config['build']

    return yaml.dump(compose_template, default_flow_style=False, allow_unicode=True, indent=2)

# Пример конфигурации Docker Compose
services = {
    'web': {
        'build': '.',
        'ports': ['8000:8000'],
        'environment': {
            'DATABASE_URL': 'postgresql://user:pass@db:5432/app',
            'DEBUG': 'True'
        },
        'depends_on': ['db', 'redis']
    },
    'db': {
        'image': 'postgres:13',
        'environment': {
            'POSTGRES_DB': 'app',
            'POSTGRES_USER': 'user',
            'POSTGRES_PASSWORD': 'pass'
        },
        'volumes': ['postgres_data:/var/lib/postgresql/data']
    },
    'redis': {
        'image': 'redis:6-alpine',
        'ports': ['6379:6379']
    }
}

docker_compose = create_docker_compose(services)
print("Docker Compose файл:")
print(docker_compose)
```

Часть 4: Автоматизация и валидация (≈4 часа)

Задача: Освоить инструменты автоматизации создания и валидации текстовых файлов.

1. Скрипт автоматической генерации документации:

```
python
import os
import glob
```

```
from pathlib import Path
from datetime import datetime

class DocumentationAutomation:
    """Автоматизация создания документации"""

    def __init__(self, project_root: str):
        self.project_root = Path(project_root)
        self.docs_dir = self.project_root / 'docs'
        self.docs_dir.mkdir(exist_ok=True)

    def generate_file_structure(self) -> str:
        """Генерация структуры файлов проекта"""
        structure = "# Структура проекта\n\n"
        structure += "```\n"

        for file_path in self.project_root.rglob('*'):
            if file_path.is_dir():
                continue

            relative_path = file_path.relative_to(self.project_root)
            depth = len(relative_path.parts) - 1
            indent = '  ' * depth

            structure += f"{indent}{relative_path.name}\n"

        structure += "```\n"
        return structure

    def create_contributing_guide(self) -> str:
        """Создание руководства для контрибьюторов"""
        guide = """# Руководство для контрибьюторов

Спасибо за интерес к contribution в наш проект!

## Процесс разработки

1. Форкните репозиторий
2. Создайте feature branch (`git checkout -b feature/amazing-feature`)
3. Commit изменений (`git commit -m 'Add amazing feature'`)
4. Push в branch (`git push origin feature/amazing-feature`)
5. Откройте Pull Request

## Стандарты кода

### Python
- Используйте Black для форматирования
- Соблюдайте PEP 8
- Пишите docstrings для всех функций

### Документация
- Обновляйте README при изменениях
- Добавляйте комментарии к новому коду
- Обновляйте CHANGELOG.md

## Тестирование

Запуск тестов:
```bash
pytest tests/
**Вопросы**
Если у вас есть вопросы, создайте issue или свяжитесь с maintainers.
"""
return guide
text
def create_license_file(self, license_type: str = "MIT") -> str:
    """Создание файла лицензии"""
    licenses = {
        "MIT": """MIT License
Copyright (c) 2024 {project_name}
Разрешается бесплатно, любым лицам, получить копию данного программного
обеспечения и сопутствующей документации (далее "Программное обеспечение"),
...
Полный текст лицензии: https://opensource.org/licenses/MIT
""",
"Apache-2.0": """Apache License 2.0
```
```

```
Copyright 2024 {project_name}
...
"""
}
text
    return licenses.get(license_type, licenses["MIT"]).format(
        project_name=self.project_root.name
    )

def generate_complete_documentation(self):
    """Генерация полного набора документации"""
    docs = {
        'README.md': self.generate_file_structure(),
        'CONTRIBUTING.md': self.create_contributing_guide(),
        'LICENSE': self.create_license_file(),
        'CHANGELOG.md': TextFileTemplates.create_changelog()
    }

    for filename, content in docs.items():
        file_path = self.docs_dir / filename
        with open(file_path, 'w', encoding='utf-8') as f:
            f.write(content)
        print(f"Создан файл: {file_path}")

def validate_markdown_files(self) -> Dict[str, List[str]]:
    """Простая валидация Markdown файлов"""
    issues = {}

    for md_file in self.docs_dir.glob('*.md'):
        with open(md_file, 'r', encoding='utf-8') as f:
            content = f.read()

        file_issues = []

        # Проверка наличия заголовка
        if not content.startswith('#'):
            file_issues.append("Отсутствует заголовок первого уровня")

        # Проверка длинных строк
        for i, line in enumerate(content.split('\n'), 1):
            if len(line) > 100 and not line.startswith('```'):
                file_issues.append(f"Строка {i} слишком длинная ({len(line)} символов)")

        if file_issues:
            issues[str(md_file)] = file_issues

    return issues
```

Пример использования

```
automation = DocumentationAutomation('/path/to/project')
automation.generate_complete_documentation()
issues = automation.validate_markdown_files()
if issues:
print("Найдены проблемы в Markdown файлах:")
for file, file_issues in issues.items():
print(f"\n{file}:")
for issue in file_issues:
print(f" - {issue}")
else:
print("Все Markdown файлы прошли валидацию")
text

**2. Практическое задание "Комплексная документация":**
```python
def create_technical_specification(project_spec):
    """Создание технического задания"""

    spec_template = f"""# Техническое задание: {project_spec['name']}

## Общая информация

- **Проект:** {project_spec['name']}
- **Версия:** {project_spec.get('version', '1.0.0')}
- **Дата создания:** {datetime.now().strftime('%Y-%m-%d')}
- **Автор:** {project_spec.get('author', 'Команда разработки')}

## Цели проекта
```
```

```
{project_spec.get('goals', '')}

## Функциональные требования

### Основной функционал
{create_requirements_list(project_spec.get('core_features', []))}

### Дополнительный функционал
{create_requirements_list(project_spec.get('additional_features', []))}

## Технические требования

### Архитектура
{project_spec.get('architecture', '')}

### Технологический стек
{create_tech_stack_table(project_spec.get('tech_stack', {}))}

## План разработки

### Этап 1: Базовый функционал
- Срок: {project_spec.get('timeline', {}).get('phase1', '2 недели')}
- Задачи: {', '.join(project_spec.get('phase1_tasks', []))}

### Этап 2: Расширенный функционал
- Срок: {project_spec.get('timeline', {}).get('phase2', '3 недели')}
- Задачи: {', '.join(project_spec.get('phase2_tasks', []))}

## Критерии приемки

{create_acceptance_criteria(project_spec.get('acceptance_criteria', []))}
"""
    return spec_template

def create_requirements_list(requirements):
    """Создание списка требований"""
    if not requirements:
        return "- Требования не определены"

    return "\n".join([f"- {req}" for req in requirements])

def create_tech_stack_table(tech_stack):
    """Создание таблицы технологического стека"""
    if not tech_stack:
        return "Технологический стек не определен"

    table = "| Компонент | Технология | Версия |\n"
    table += "|-----------|------------|--------|\n"

    for component, tech in tech_stack.items():
        table += f"| {component} | {tech.get('technology', '')} | {tech.get('version', '')} |\n"

    return table

# Пример технического задания
project_spec = {
    'name': 'Система управления контентом',
    'version': '2.0.0',
    'author': 'Команда разработки ContentTech',
    'goals': 'Создание современной системы управления контентом с поддержкой многопользовательского режима и
расширенными возможностями публикации.',
    'core_features': [
        'Управление пользователями и ролями',
        'Создание и редактирование контента',
        'Публикация и планирование',
        'Управление медиа-файлами'
    ],
    'tech_stack': {
        'backend': {'technology': 'Python Django', 'version': '4.2'},
        'frontend': {'technology': 'React', 'version': '18.0'},
        'database': {'technology': 'PostgreSQL', 'version': '14.0'},
        'deployment': {'technology': 'Docker', 'version': '20.0'}
    },
    'phase1_tasks': ['Базовая аутентификация', 'CRUD операции для контента', 'Базовая админ-панель'],
    'acceptance_criteria': [
        'Система поддерживает не менее 100 одновременных пользователей',
        'Время отклика API не превышает 200 мс',
```

```
        'Поддержка русского и английского языков'
    ]
}

tech_spec = create_technical_specification(project_spec)
print("Техническое задание создано:")
print(tech_spec[:500] + "...")
```

Итоговое задание для самостоятельного выполнения:
Сценарий: Создание полного набора текстовых файлов для open-source проекта.
Задание:
1. Создание базовой документации:
python
 # Генерация основного набора файлов
base_files = {
'README.md': 'Основная документация проекта',
'CHANGELOG.md': 'История изменений',
'CONTRIBUTING.md': 'Руководство для контрибьюторов',
'CODE_OF_CONDUCT.md': 'Правила поведения',
'LICENSE': 'Лицензия проекта',
'docs/API.md': 'Документация API',
'docs/DEVELOPMENT.md': 'Руководство разработчика',
'docs/DEPLOYMENT.md': 'Инструкции по развертыванию'
}
2. Структура проекта:
text
my-awesome-project/
├── README.md
├── CHANGELOG.md
├── CONTRIBUTING.md
├── CODE_OF_CONDUCT.md
├── LICENSE
├── docs/
│ ├── API.md
│ ├── DEVELOPMENT.md
│ ├── DEPLOYMENT.md
│ └── images/
├── config/
│ ├── config.yaml
│ └── docker-compose.yml
├── scripts/
│ └── generate_docs.py
└── tests/
└── README.md
3. Требования к файлам:
README.md должен содержать:

- Заголовок и краткое описание
- Быстрый старт (quick start)
- Особенности и возможности
- Примеры использования
- Ссылки на документацию

API.md должен содержать:

- Описание всех endpoints
- Примеры запросов и ответов
- Коды ошибок
- Аутентификация и авторизация

Конфигурационные файлы:

- config.yaml с настройками приложения
- docker-compose.yml для развертывания
- .env.example с переменными окружения

4. Автоматизация:
python
 # Скрипт для автоматической генерации и валидации
def create_project_documentation(project_config):
"""Создание полной документации проекта"""

automation = DocumentationAutomation(project_config['root_path'])

 # Генерация базовых файлов
automation.generate_complete_documentation()

 # Создание специфических файлов
additional_files = {
'API.md': create_api_documentation(project_config['api']),
'DEVELOPMENT.md': create_development_guide(project_config),
'DEPLOYMENT.md': create_deployment_guide(project_config)

```
    }

    for filename, content in additional_files.items():
        file_path = automation.docs_dir / filename
        with open(file_path, 'w', encoding='utf-8') as f:
            f.write(content)

    # Валидация
    issues = automation.validate_markdown_files()

    return {
        'generated_files': list(automation.docs_dir.glob('*.md')),
        'validation_issues': issues
    }
```

5. Критерии оценки:

Компонент	Критерии
Полнота документации	Наличие всех необходимых файлов
Качество содержания	Информативность, точность, актуальн
Структура и форматирование	Соблюдение стандартов, читаемость
Автоматизация	Скрипты генерации и валидации
Профессиональное оформление	Единый стиль, грамотность

Это занятие даст студентам практические навыки создания профессиональной текстовой документации, что является ключевой компетенцией для разработчиков, технических писателей и DevOps-инженеров.
Практическое занятие 79 - 82. Разработка руководства поль-зователя несложного программного средства или краткого руководства по эксплуатации несложного технического средства
Отлично! Разработаем комплексный план практического занятия по созданию руководства пользователя.

Практическое занятие 79-82: Разработка руководства пользователя программного средства
Цель занятия: Освоить методику создания понятных и эффективных руководств пользователя для программных продуктов и технических средств.
Программное обеспечение:

- **Документирование:** Microsoft Word, Google Docs, MadCap Flare
- **Скриншоты:** Snagit, Greenshot, Lightshot
- **Диаграммы:** Draw.io, Lucidchart, Visio
- **Версионность:** Git, GitHub, GitLab
- **Публикация:** PDF, HTML, CHM

Необходимые знания: Базовые навыки технического письма, понимание принципов UX.

План занятия и Ход работы
Часть 1: Анализ продукта и аудитории (≈4 часа)
Задача: Провести анализ программного средства и определить целевую аудиторию.
1. Шаблон анализа продукта:
python

```
class ProductAnalyzer:
    """Анализ программного средства для документации"""

    def __init__(self, product_name, version):
        self.product_name = product_name
        self.version = version
        self.features = []
        self.audience = []

    def analyze_features(self):
        """Анализ функциональных возможностей"""
        features = {
            'basic_operations': [
                'Установка и настройка',
                'Запуск приложения',
                'Основной интерфейс',
                'Базовые операции'
            ],
            'advanced_features': [
```

```
                'Настройка параметров',
                'Расширенные функции',
                'Интеграции',
                'Автоматизация'
            ],
            'maintenance': [
                'Обновление',
                'Резервное копирование',
                'Устранение неполадок',
                'Оптимизация'
            ]
        }
        return features

    def define_audience(self):
        """Определение целевой аудитории"""
        audience_types = {
            'novice_users': {
                'description': 'Новые пользователи без опыта',
                'needs': ['Пошаговые инструкции', 'Базовые понятия', 'Примеры'],
                'technical_level': 'Низкий'
            },
            'regular_users': {
                'description': 'Пользователи с базовым опытом',
                'needs': ['Эффективные методы', 'Советы', 'Решение проблем'],
                'technical_level': 'Средний'
            },
            'power_users': {
                'description': 'Опытные пользователи',
                'needs': ['Расширенные функции', 'API документация', 'Оптимизация'],
                'technical_level': 'Высокий'
            }
        }
        return audience_types

    def create_user_personas(self):
        """Создание персонажей пользователей"""
        personas = [
            {
                'name': 'Анна, менеджер',
                'role': 'Бизнес-пользователь',
                'goals': ['Быстро освоить базовые функции', 'Эффективно выполнять задачи'],
                'frustrations': ['Сложный интерфейс', 'Непонятные термины'],
                'tech_skills': 'Базовые'
            },
            {
                'name': 'Иван, разработчик',
                'role': 'Технический специалист',
                'goals': ['Интегрировать с другими системами', 'Автоматизировать процессы'],
                'frustrations': ['Недостаток технической информации', 'Сложная настройка'],
                'tech_skills': 'Продвинутые'
            }
        ]
        return personas

# Пример использования
analyzer = ProductAnalyzer("TextProcessor Pro", "2.1.0")
features = analyzer.analyze_features()
audience = analyzer.define_audience()
personas = analyzer.create_user_personas()

print("Анализ продукта завершен:")
print(f"Функции: {len(features['basic_operations']) + len(features['advanced_features'])}")
print(f"Типы пользователей: {len(audience)}")
```

2. Практическое задание "Анализ мобильного приложения":

```
markdown
### Продукт: Мобильное приложение "FinanceTracker"

**Основные функции:**
- Учет доходов и расходов
- Создание бюджетов
- Визуализация финансов
- Напоминания о платежах

**Целевая аудитория:**
1. **Студенты** - простой учет карманных денег
2. **Фрилансеры** - управление нерегулярными доходами
```

3. **Семьи** - совместный бюджет и планирование

Задание: Разработать структуру руководства для каждой группы пользователей

Часть 2: Структура руководства пользователя (≈4 часа)
Задача: Создать эффективную структуру руководства.
1. Шаблон руководства пользователя:
python
```
class UserManualTemplate:
    """Шаблон руководства пользователя"""

    def __init__(self, product_name):
        self.product_name = product_name
        self.structure = []

    def create_standard_structure(self):
        """Стандартная структура руководства"""
        structure = [
            {
                'section': 'Титульная страница',
                'subsections': [
                    'Название продукта и версия',
                    'Дата публикации',
                    'Контактная информация'
                ]
            },
            {
                'section': 'Введение',
                'subsections': [
                    'О продукте',
                    'Целевая аудитория',
                    'Структура руководства',
                    'Соглашения и обозначения'
                ]
            },
            {
                'section': 'Быстрый старт',
                'subsections': [
                    'Минимальные требования',
                    'Установка',
                    'Первая настройка',
                    'Выполнение первой задачи'
                ]
            },
            {
                'section': 'Основные функции',
                'subsections': [
                    'Обзор интерфейса',
                    'Работа с данными',
                    'Создание и сохранение',
                    'Базовые операции'
                ]
            },
            {
                'section': 'Расширенные возможности',
                'subsections': [
                    'Настройка параметров',
                    'Интеграции',
                    'Автоматизация',
                    'Экспорт данных'
                ]
            },
            {
                'section': 'Решения проблем',
                'subsections': [
                    'Частые вопросы (FAQ)',
                    'Коды ошибок',
                    'Восстановление данных',
                    'Контакты поддержки'
                ]
            },
            {
                'section': 'Приложения',
                'subsections': [
                    'Горячие клавиши',
                    'Спецификации',
```

```
                'Глоссарий',
                'Индекс'
            ]
        }
    ]
    return structure

def generate_toc(self):
    """Генерация оглавления"""
    toc = f"# Руководство пользователя: {self.product_name}\n\n"
    toc += "## Оглавление\n\n"

    for i, section in enumerate(self.structure, 1):
        toc += f"{i}. **{section['section']}**\n"
        for j, subsection in enumerate(section['subsections'], 1):
            toc += f"   {i}.{j} {subsection}\n"
        toc += "\n"

    return toc

# Создание структуры для приложения учета финансов
finance_template = UserManualTemplate("FinanceTracker")
finance_template.structure = finance_template.create_standard_structure()

print("Структура руководства:")
print(finance_template.generate_toc())
```

2. Практическое задание "Создание раздела":

```
markdown
### Раздел: "Быстрый старт с FinanceTracker"

**Цель:** Помочь новому пользователю быстро начать работу

**Содержание:**
1. **Установка приложения**
   - Скачивание из App Store/Google Play
   - Разрешения и настройки

2. **Первая настройка**
   - Создание профиля
   - Настройка валюты
   - Выбор категорий расходов

3. **Добавление первой операции**
   - Доход: стипендия 15000 ₽
   - Расход: продукты 2500 ₽

4. **Просмотр статистики**
   - Основной экран
   - Быстрый обзор финансов
```

Часть 3: Создание пошаговых инструкций (≈4 часа)

Задача: Научиться писать четкие и понятные инструкции.

1. Генератор инструкций:

```
python
class InstructionBuilder:
    """Построитель пошаговых инструкций"""

    def __init__(self):
        self.steps = []
        self.screenshots = []

    def add_step(self, step_number, action, description, expected_result=None):
        """Добавление шага инструкции"""
        step = {
            'number': step_number,
            'action': action,
            'description': description,
            'expected_result': expected_result,
            'tip': None,
            'warning': None
        }
        self.steps.append(step)

    def add_tip(self, step_number, tip_text):
        """Добавление подсказки к шагу"""
        for step in self.steps:
```

```
            if step['number'] == step_number:
                step['tip'] = tip_text
                break

    def add_warning(self, step_number, warning_text):
        """Добавление предупреждения к шагу"""
        for step in self.steps:
            if step['number'] == step_number:
                step['warning'] = warning_text
                break

    def generate_instructions(self, task_name):
        """Генерация полной инструкции"""
        instructions = f"## {task_name}\n\n"

        for step in self.steps:
            instructions += f"### Шаг {step['number']}: {step['action']}\n\n"
            instructions += f"{step['description']}\n\n"

            if step['expected_result']:
                instructions += f"**Ожидаемый результат:** {step['expected_result']}\n\n"

            if step['tip']:
                instructions += f"> 💡 **Совет:** {step['tip']}\n\n"

            if step['warning']:
                instructions += f"> ⚠️ **Внимание:** {step['warning']}\n\n"

        return instructions

# Пример создания инструкции по добавлению операции
instruction_builder = InstructionBuilder()

instruction_builder.add_step(
    1,
    "Откройте главный экран",
    "Запустите приложение FinanceTracker. Нажмите на иконку '+' в нижней части экрана.",
    "Откроется форма добавления новой операции"
)

instruction_builder.add_step(
    2,
    "Выберите тип операции",
    "Нажмите на переключатель 'Тип операции' и выберите 'Расход' или 'Доход'",
    "Поле типа операции изменит цвет"
)

instruction_builder.add_tip(2, "Для быстрого выбора проведите пальцем влево/вправо")

instruction_builder.add_step(
    3,
    "Введите сумму",
    "Введите сумму операции с помощью цифровой клавиатуры",
    "Сумма отобразится в поле ввода"
)

instruction_builder.add_warning(3, "Проверьте правильность ввода суммы перед сохранением")

instruction_builder.add_step(
    4,
    "Выберите категорию",
    "Нажмите на поле 'Категория' и выберите подходящую категорию из списка",
    "Категория будет назначена операции"
)

instruction_builder.add_step(
    5,
    "Сохраните операцию",
    "Нажмите кнопку 'Сохранить' в правом верхнем углу экрана",
    "Операция добавится в общий список и обновит статистику"
)

instructions = instruction_builder.generate_instructions("Добавление финансовой операции")
print(instructions)
```

2. Практическое задание "Инструкция по созданию бюджета":

python

```
def create_budget_instruction():
```

```
    """Создание инструкции по созданию бюджета"""

    builder = InstructionBuilder()

    steps = [
        (1, "Перейдите в раздел 'Бюджеты'", "На главном экране нажмите на иконку 'Бюджет' в нижнем меню"),
        (2, "Создайте новый бюджет", "Нажмите кнопку '+' в правом верхнем углу"),
        (3, "Заполните параметры", "Введите название бюджета, общую сумму и период"),
        (4, "Добавьте категории", "Выберите категории расходов для включения в бюджет"),
        (5, "Настройте лимиты", "Для каждой категории установите месячный лимит"),
        (6, "Сохраните бюджет", "Нажмите 'Готово' для сохранения настроек")
    ]

    for step in steps:
        builder.add_step(step[0], step[1], step[2])

    builder.add_tip(3, "Используйте понятные названия, например 'Бюджет на январь 2024'")
    builder.add_tip(4, "Включайте только те категории, которые планируете контролировать")
    builder.add_warning(5, "Устанавливайте реалистичные лимиты на основе прошлых расходов")

    return builder.generate_instructions("Создание месячного бюджета")

budget_guide = create_budget_instruction()
print(budget_guide)
```

Часть 4: Визуальное оформление и скриншоты (≈4 часа)
Задача: Научиться эффективно использовать визуальные элементы.
1. Система аннотаций для скриншотов:
python

```
class ScreenshotManager:
    """Управление скриншотами для документации"""

    def __init__(self):
        self.screenshots = []
        self.annotations = []

    def add_screenshot(self, filename, description, context):
        """Добавление информации о скриншоте"""
        screenshot = {
            'filename': filename,
            'description': description,
            'context': context,
            'annotations': [],
            'alt_text': f"Скриншот: {description}"
        }
        self.screenshots.append(screenshot)
        return len(self.screenshots) - 1

    def add_annotation(self, screenshot_id, element, description):
        """Добавление аннотации к скриншоту"""
        annotation = {
            'element': element,
            'description': description,
            'position': 'auto'  # auto, top, bottom, left, right
        }
        self.screenshots[screenshot_id]['annotations'].append(annotation)

    def generate_screenshot_guide(self):
        """Генерация руководства по скриншотам"""
        guide = "# Визуальное руководство\n\n"

        for i, screenshot in enumerate(self.screenshots, 1):
            guide += f"## Рисунок {i}: {screenshot['description']}\n\n"
            guide += f"![{screenshot['alt_text']}]({screenshot['filename']})\n\n"
            guide += f"*Контекст: {screenshot['context']}*\n\n"

            if screenshot['annotations']:
                guide += "**Элементы интерфейса:**\n"
                for annotation in screenshot['annotations']:
                    guide += f"- **{annotation['element']}**: {annotation['description']}\n"
                guide += "\n"

        return guide

# Пример использования
screenshot_manager = ScreenshotManager()
```

```
# Главный экран
main_screen_id = screenshot_manager.add_screenshot(
    "images/main_screen.png",
    "Главный экран FinanceTracker",
    "Основной экран приложения после запуска"
)

screenshot_manager.add_annotation(main_screen_id, "Баланс", "Общий баланс счета")
screenshot_manager.add_annotation(main_screen_id, "Быстрое добавление", "Кнопка для быстрого добавления операции")
screenshot_manager.add_annotation(main_screen_id, "Навигация", "Основное меню навигации")

# Экран добавления операции
add_operation_id = screenshot_manager.add_screenshot(
    "images/add_operation.png",
    "Форма добавления операции",
    "Экран для ввода новой финансовой операции"
)

screenshot_manager.add_annotation(add_operation_id, "Тип операции", "Переключатель между доходом и расходом")
screenshot_manager.add_annotation(add_operation_id, "Сумма", "Поле для ввода суммы")
screenshot_manager.add_annotation(add_operation_id, "Категория", "Выбор категории операции")

screenshot_guide = screenshot_manager.generate_screenshot_guide()
print(screenshot_guide)
```

2. Практическое задание "Создание визуального руководства":

```
markdown
### Визуальное руководство по отчетам

**Цель:** Показать как работать с отчетами в FinanceTracker

**Скриншоты:**
1. **Экран отчетов** - общий вид раздела
2. **Круговая диаграмма расходов** - визуализация по категориям
3. **График динамики** - изменения баланса за период
4. **Детальный отчет** - таблица с операциями

**Аннотации для каждого скриншота:**
- Кнопки фильтрации по дате
- Элементы настройки отображения
- Кнопки экспорта данных
- Подписи осей и легенды

**Результат:** PDF файл с 4 аннотированными скриншотами и пояснениями
```

Часть 5: Сборка и публикация (≈4 часа)

Задача: Научиться собирать руководство в финальный формат.

1. Система сборки документации:

```
python
class DocumentationBuilder:
    """Сборщик финальной документации"""

    def __init__(self, product_name, version):
        self.product_name = product_name
        self.version = version
        self.sections = []
        self.metadata = {}

    def add_section(self, title, content, level=1):
        """Добавление раздела в документацию"""
        self.sections.append({
            'title': title,
            'content': content,
            'level': level
        })

    def generate_manual(self):
        """Генерация полного руководства"""
        manual = f"""# Руководство пользователя\n# {self.product_name}\n\n"""
        manual += f"**Версия:** {self.version}  \n"
        manual += f"**Дата публикации:** 15 января 2024 года\n\n"

        manual += "---\n\n"

        # Оглавление
        manual += "## Оглавление\n\n"
```

```
        for i, section in enumerate(self.sections, 1):
            indent = "  " * (section['level'] - 1)
            manual += f"{indent}{i}. [{section['title']}](#{section['title'].lower().replace(' ', '-')})\n"
        manual += "\n---\n\n"

        # Содержание
        for i, section in enumerate(self.sections, 1):
            hashes = "#" * (section['level'] + 1)
            manual += f"{hashes} {i}. {section['title']}\n\n"
            manual += section['content'] + "\n\n"

        return manual

    def export_to_pdf(self, filename):
        """Экспорт в PDF (имитация)"""
        content = self.generate_manual()

        # В реальной реализации здесь был бы вызов библиотеки для генерации PDF
        print(f"Генерация PDF: {filename}")
        print(f"Размер контента: {len(content)} символов")
        print("    PDF файл успешно создан")

        return {
            'filename': filename,
            'size': len(content),
            'sections': len(self.sections)
        }

    def create_quick_start_guide(self):
        """Создание краткого руководства"""
        quick_guide = f"# Краткое руководство: {self.product_name}\n\n"
        quick_guide += "## Основные шаги для начала работы:\n\n"

        basic_steps = [
            "1. **Установите приложение** из официального магазина",
            "2. **Зарегистрируйтесь** или войдите в существующий аккаунт",
            "3. **Настройте основные параметры**: валюта, категории",
            "4. **Добавьте первую операцию** для тестирования",
            "5. **Ознакомьтесь с отчетом** для понимания возможностей"
        ]

        quick_guide += "\n".join(basic_steps)
        quick_guide += "\n\n## Полезные советы:\n\n"
        quick_guide += "- Регулярно вносите операции для точной статистики\n"
        quick_guide += "- Используйте бюджеты для контроля расходов\n"
        quick_guide += "- Настройте уведомления о важных событиях\n"

        return quick_guide

# Пример сборки полного руководства
doc_builder = DocumentationBuilder("FinanceTracker", "2.1.0")

# Добавление разделов
doc_builder.add_section("Введение", """
FinanceTracker - это мобильное приложение для учета личных финансов.
С помощью этого приложения вы можете:

- Отслеживать доходы и расходы
- Планировать бюджеты
- Анализировать финансовые привычки
- Контролировать достижение финансовых целей
""")

doc_builder.add_section("Быстрый старт", """
Для начала работы выполните следующие шаги:

1. Установите приложение из App Store или Google Play
2. Запустите приложение и создайте учетную запись
3. Настройте основные параметры в разделе "Настройки"
4. Добавьте свои первые финансовые операции
""")

doc_builder.add_section("Основные функции", create_budget_instruction(), level=2)

# Генерация и экспорт
full_manual = doc_builder.generate_manual()
quick_guide = doc_builder.create_quick_start_guide()
```

```
print("Полное руководство создано:")
print(f"Разделы: {len(doc_builder.sections)}")
print(f"Объем: {len(full_manual)} символов")

print("\nКраткое руководство:")
print(quick_guide[:200] + "...")
```
2. Практическое задание "Финальная сборка":
```
python
def create_complete_user_manual():
    """Создание полного руководства пользователя"""

    manual_builder = DocumentationBuilder("FinanceTracker Mobile", "2.1.0")

    # Добавление всех разделов
    sections = [
        ("Введение", """
FinanceTracker помогает вам контролировать личные финансы в удобном мобильном формате.
Это руководство описывает все функции приложения и предоставляет пошаговые инструкции.
"""),

        ("Установка и настройка", """
### Установка приложения
- App Store (iOS)
- Google Play (Android)

### Первый запуск
- Регистрация аккаунта
- Базовая настройка
- Синхронизация данных
"""),

        ("Работа с операциями", """
Добавление, редактирование и удаление финансовых операций.
Включает доходы, расходы и переводы между счетами.
"""),

        ("Бюджеты и планирование", """
Создание месячных бюджетов, отслеживание выполнения, настройка категорий.
"""),

        ("Отчеты и аналитика", """
Генерация отчетов, графики, экспорт данных, анализ финансовых привычек.
"""),

        ("Частые вопросы", """
Решения распространенных проблем, контакты поддержки, дополнительные ресурсы.
""")
    ]

    for title, content in sections:
        manual_builder.add_section(title, content)

    # Экспорт в различные форматы
    results = {
        'full_manual': manual_builder.generate_manual(),
        'quick_guide': manual_builder.create_quick_start_guide(),
        'pdf_export': manual_builder.export_to_pdf("FinanceTracker_Manual.pdf")
    }

    return results

# Запуск создания руководства
final_results = create_complete_user_manual()
print("Руководство пользователя создано!")
print(f"Форматы: PDF, Online, Quick Guide")
print(f"Общий объем документации: {len(final_results['full_manual'])} символов")
```

Итоговое задание для самостоятельного выполнения:
Сценарий: Создание полного руководства пользователя для мобильного приложения "HealthTracker".
Задание:
1. Описание продукта:
```
python
health_tracker_spec = {
    'name': 'HealthTracker Pro',
    'version': '1.0.0',
```

```
    'description': 'Приложение для отслеживания здоровья и фитнеса',
    'main_features': [
        'Учет физической активности',
        'Контроль питания и калорий',
        'Мониторинг сна',
        'Создание тренировок',
        'Напоминания и цели'
    ],
    'target_audience': [
        'Люди, ведущие здоровый образ жизни',
        'Спортсмены',
        'Люди с медицинскими показаниями',
        'Фитнес-тренеры'
    ]
}
```

2. Структура руководства:

text

```
HealthTracker_Manual/
├── README.md
├── User_Manual_Full.md
├── Quick_Start_Guide.md
├── images/
│   ├── main_interface.png
│   ├── add_workout.png
│   ├── nutrition_tracker.png
│   └── sleep_analysis.png
├── tutorials/
│   ├── first_setup.md
│   ├── workout_creation.md
│   └── goal_setting.md
└── troubleshooting/
    ├── faq.md
    └── common_issues.md
```

3. Требования к содержанию:

Полное руководство должно включать:

- Подробное описание всех функций
- Пошаговые инструкции с скриншотами
- Примеры использования для разных сценариев
- Решения частых проблем

Краткое руководство должно содержать:

- Быстрый старт за 15 минут
- Основные операции "must-know"
- Полезные советы для новичков
- Контакты поддержки

4. Критерии оценки:

Компонент	Критерии
Полнота и точность	Описание всех функций, корректность информации
Структура и организация	Логичная последовательность, удобная навигация
Ясность и доступность	Понятный язык, соответствие целевой аудитории
Визуальные материалы	Качество скриншотов, уместность аннотаций
Практическая ценность	Полезность инструкций, решение реальных задач

5. Дополнительные требования:

- Руководство должно быть доступно в 2-х форматах: PDF и онлайн
- Не более 50 страниц для полной версии
- Не более 10 страниц для краткой версии
- Использование единого стиля оформления
- Проверка грамотности и терминологии

Это занятие даст студентам практические навыки создания профессиональных руководств пользователя, которые являются ключевым элементом успешного программного продукта и важной компетенцией для технических писателей и продуктовых дизайнеров.

Практическое занятие 83 - 86. Анализ замечаний экспертов и внесение исправлений в документ

Отлично! Разработаем комплексный план практического занятия по анализу замечаний экспертов и внесению исправлений.

Практическое занятие 83-86: Анализ замечаний экспертов и внесение исправлений в документ
Цель занятия: Освоить методику систематического анализа экспертных замечаний, приоритизации правок и эффективного внесения изменений в технические документы.
Программное обеспечение:

- **Редакторы:** Microsoft Word (режим рецензирования), Google Docs (предложения), VS Code
- **Системы контроля версий:** Git, GitHub, GitLab
- **Инструменты сравнения:** DiffChecker, Beyond Compare, WinMerge
- **Управление задачами:** Trello, Asana, Jira
- **Коммуникация:** Slack, Microsoft Teams, Email

Необходимые знания: Базовые навыки технического письма, работа с системами контроля версий.

План занятия и Ход работы
Часть 1: Систематизация экспертных замечаний (≈4 часа)
Задача: Научиться структурировать и категоризировать замечания экспертов.
1. Классификатор замечаний:
python

```
class FeedbackAnalyzer:
    """Анализ и систематизация экспертных замечаний"""

    def __init__(self, document_name, document_version):
        self.document_name = document_name
        self.document_version = document_version
        self.feedback_items = []
        self.categories = {}

    def add_feedback(self, expert_name, section, comment, feedback_type, priority="medium"):
        """Добавление замечания эксперта"""
        feedback_item = {
            'id': len(self.feedback_items) + 1,
            'expert': expert_name,
            'section': section,
            'comment': comment,
            'type': feedback_type,
            'priority': priority,
            'status': 'new',
            'action_required': True,
            'assigned_to': None,
            'resolution': None,
            'date_added': '2024-01-15'
        }
        self.feedback_items.append(feedback_item)
        return feedback_item['id']

    def categorize_feedback(self):
        """Автоматическая категоризация замечаний"""
        categories = {
            'technical_accuracy': {
                'description': 'Техническая точность и корректность',
                'items': [],
                'count': 0
            },
            'clarity': {
                'description': 'Ясность и понятность изложения',
                'items': [],
                'count': 0
            },
            'completeness': {
                'description': 'Полнота информации',
                'items': [],
                'count': 0
            },
            'structure': {
                'description': 'Структура и организация документа',
                'items': [],
                'count': 0
            },
            'formatting': {
                'description': 'Форматирование и оформление',
                'items': [],
                'count': 0
            },
            'language': {
                'description': 'Язык и терминология',
                'items': [],
                'count': 0
```

```
            }
        }

        for item in self.feedback_items:
            if 'техни' in item['comment'].lower() or 'точн' in item['comment'].lower():
                categories['technical_accuracy']['items'].append(item)
            elif 'понят' in item['comment'].lower() or 'ясн' in item['comment'].lower():
                categories['clarity']['items'].append(item)
            elif 'полно' in item['comment'].lower() or 'недостает' in item['comment'].lower():
                categories['completeness']['items'].append(item)
            elif 'структур' in item['comment'].lower() or 'организац' in item['comment'].lower():
                categories['structure']['items'].append(item)
            elif 'формат' in item['comment'].lower() or 'оформлен' in item['comment'].lower():
                categories['formatting']['items'].append(item)
            else:
                categories['language']['items'].append(item)

        # Подсчет количества
        for category in categories:
            categories[category]['count'] = len(categories[category]['items'])

        self.categories = categories
        return categories

    def generate_feedback_report(self):
        """Генерация отчета по замечаниям"""
        self.categorize_feedback()

        report = f"# Отчет по экспертных замечаниям\n\n"
        report += f"**Документ:** {self.document_name}\n"
        report += f"**Версия:** {self.document_version}\n"
        report += f"**Всего замечаний:** {len(self.feedback_items)}\n"
        report += f"**Дата анализа:** 2024-01-15\n\n"

        report += "## Распределение по категориям:\n\n"
        for category_name, category_data in self.categories.items():
            report += f"- **{category_data['description']}:** {category_data['count']} замечаний\n"

        report += "\n## Приоритетные замечания:\n\n"
        high_priority = [item for item in self.feedback_items if item['priority'] == 'high']
        for item in high_priority:
            report += f"###     {item['expert']} - {item['section']}\n"
            report += f"**Замечание:** {item['comment']}\n"
            report += f"**Тип:** {item['type']}\n\n"

        return report

# Пример использования
analyzer = FeedbackAnalyzer("Руководство пользователя FinanceTracker", "2.1.0")

# Добавление замечаний экспертов
analyzer.add_feedback(
    "Иван Петров",
    "Раздел 'Установка'",
    "Не указаны минимальные системные требования для Android 8.0",
    "completeness",
    "high"
)

analyzer.add_feedback(
    "Мария Сидорова",
    "Глава 3, страница 15",
    "Термин 'бюджетирование' используется непоследовательно, иногда 'планирование бюджета'",
    "language",
    "medium"
)

analyzer.add_feedback(
    "Алексей Козлов",
    "Инструкция по экспорту данных",
    "Скриншоты устарели - интерфейс изменился в версии 2.0",
    "technical_accuracy",
    "high"
)

report = analyzer.generate_feedback_report()
print(report)
```

2. Практическое задание "Анализ реальных замечаний":

```
python
# Замечания от экспертов по техническому документу
expert_feedback = [
    {
        'expert': 'Технический руководитель',
        'section': 'Архитектура системы',
        'comment': 'Диаграмма не отражает новые компоненты мониторинга',
        'type': 'completeness',
        'priority': 'high'
    },
    {
        'expert': 'Lead Developer',
        'section': 'API документация',
        'comment': 'Отсутствуют примеры ошибок для endpoint /api/v1/users',
        'type': 'completeness',
        'priority': 'medium'
    },
    {
        'expert': 'Технический писатель',
        'section': 'Введение',
        'comment': 'Слишком сложные предложения, нужно упростить для новой аудитории',
        'type': 'clarity',
        'priority': 'medium'
    },
    {
        'expert': 'QA инженер',
        'section': 'Установка и настройка',
        'comment': 'Шаги 5-7 не воспроизводятся в тестовой среде',
        'type': 'technical_accuracy',
        'priority': 'high'
    }
]

def analyze_expert_feedback(feedback_list):
    """Анализ списка замечаний от экспертов"""
    analyzer = FeedbackAnalyzer("Техническое задание системы", "1.2.0")

    for feedback in feedback_list:
        analyzer.add_feedback(
            feedback['expert'],
            feedback['section'],
            feedback['comment'],
            feedback['type'],
            feedback['priority']
        )

    return analyzer.generate_feedback_report()

analysis_result = analyze_expert_feedback(expert_feedback)
print(analysis_result)
```

Часть 2: Приоритизация и планирование исправлений (≈4 часа)
Задача: Научиться определять приоритеты и планировать работу по исправлениям.
1. Матрица приоритизации:

```
python
class CorrectionPlanner:
    """Планирование исправлений на основе замечаний"""

    def __init__(self, feedback_analyzer):
        self.analyzer = feedback_analyzer
        self.action_plan = []
        self.timeline = {}

    def assess_impact(self, feedback_item):
        """Оценка влияния замечания на качество документа"""
        impact_scores = {
            'technical_accuracy': 9,
            'completeness': 7,
            'clarity': 6,
            'structure': 5,
            'language': 4,
            'formatting': 3
        }

        priority_scores = {
```

```
            'high': 10,
            'medium': 5,
            'low': 2
        }

        type_score = impact_scores.get(feedback_item['type'], 5)
        priority_score = priority_scores.get(feedback_item['priority'], 5)

        return (type_score + priority_score) / 2

    def estimate_effort(self, feedback_item):
        """Оценка усилий для исправления"""
        effort_indicators = {
            'simple_typo': 1,
            'formatting_issue': 2,
            'clarification_needed': 3,
            'content_addition': 4,
            'structural_change': 6,
            'technical_correction': 8,
            'major_rewrite': 10
        }

        comment = feedback_item['comment'].lower()
        if 'опечатка' in comment or 'орфограф' in comment:
            return effort_indicators['simple_typo']
        elif 'формат' in comment:
            return effort_indicators['formatting_issue']
        elif 'структур' in comment:
            return effort_indicators['structural_change']
        elif 'техни' in comment:
            return effort_indicators['technical_correction']
        else:
            return effort_indicators['clarification_needed']

    def create_action_plan(self):
        """Создание плана действий по исправлениям"""
        for item in self.analyzer.feedback_items:
            impact = self.assess_impact(item)
            effort = self.estimate_effort(item)
            priority_score = impact / effort  # ROI подхода

            action_item = {
                'feedback_id': item['id'],
                'section': item['section'],
                'description': item['comment'],
                'impact': impact,
                'effort': effort,
                'priority_score': round(priority_score, 2),
                'assigned_to': self.assign_responsibility(item),
                'estimated_time': f"{effort * 30} минут",  # 30 минут за единицу усилия
                'deadline': self.calculate_deadline(priority_score),
                'status': 'planned'
            }

            self.action_plan.append(action_item)

        # Сортировка по приоритету
        self.action_plan.sort(key=lambda x: x['priority_score'], reverse=True)
        return self.action_plan

    def assign_responsibility(self, feedback_item):
        """Назначение ответственного за исправление"""
        if feedback_item['type'] in ['technical_accuracy', 'completeness']:
            return 'Технический писатель + Разработчик'
        elif feedback_item['type'] in ['clarity', 'language']:
            return 'Технический писатель'
        elif feedback_item['type'] in ['formatting', 'structure']:
            return 'Редактор'
        else:
            return 'Технический писатель'

    def calculate_deadline(self, priority_score):
        """Расчет дедлайна на основе приоритета"""
        if priority_score > 2.0:
            return '2024-01-17'  # Срочно
        elif priority_score > 1.0:
            return '2024-01-20'  # Средний приоритет
```

```
        else:
            return '2024-01-25'  # Низкий приоритет

    def generate_implementation_plan(self):
        """Генерация плана реализации исправлений"""
        plan = "# План исправлений документа\n\n"

        plan += "## Приоритетные задачи (к выполнению в первую очередь):\n\n"
        high_priority = [item for item in self.action_plan if item['priority_score'] > 1.5]
        for item in high_priority:
            plan += f"### {item['section']}\n"
            plan += f"- **Задача:** {item['description']}\n"
            plan += f"- **Приоритет:** {item['priority_score']}\n"
            plan += f"- **Ответственный:** {item['assigned_to']}\n"
            plan += f"- **Срок:** {item['deadline']}\n\n"

        plan += "## Остальные задачи:\n\n"
        other_tasks = [item for item in self.action_plan if item['priority_score'] <= 1.5]
        for item in other_tasks:
            plan += f"- {item['section']} ({item['assigned_to']}) - {item['deadline']}\n"

        return plan

# Пример использования
planner = CorrectionPlanner(analyzer)
action_plan = planner.create_action_plan()
implementation_plan = planner.generate_implementation_plan()

print(implementation_plan)
```

2. Практическое задание "Создание roadmap исправлений":

```
python
def create_correction_roadmap(feedback_data, sprint_duration=5):
    """Создание roadmap исправлений по спринтам"""

    planner = CorrectionPlanner(feedback_data)
    action_plan = planner.create_action_plan()

    # Группировка по спринтам
    sprints = {}
    current_sprint = 1
    current_sprint_effort = 0
    max_sprint_effort = 20  # макс усилий за спринт

    for task in action_plan:
        if current_sprint_effort + task['effort'] <= max_sprint_effort:
            if current_sprint not in sprints:
                sprints[current_sprint] = []
            sprints[current_sprint].append(task)
            current_sprint_effort += task['effort']
        else:
            current_sprint += 1
            current_sprint_effort = task['effort']
            if current_sprint not in sprints:
                sprints[current_sprint] = []
            sprints[current_sprint].append(task)

    # Генерация roadmap
    roadmap = "# Roadmap исправлений документа\n\n"

    for sprint_num, tasks in sprints.items():
        roadmap += f"## Спринт {sprint_num} (Дней: {sprint_duration})\n\n"

        total_effort = sum(task['effort'] for task in tasks)
        roadmap += f"**Общая сложность:** {total_effort} единиц усилия\n\n"

        for task in tasks:
            roadmap += f"- **{task['section']}** (сложность: {task['effort']})\n"
            roadmap += f"  - {task['description'][:100]}...\n"
            roadmap += f"  - Ответственный: {task['assigned_to']}\n\n"

    return roadmap

roadmap = create_correction_roadmap(analyzer)
print(roadmap)
```

Часть 3: Внесение исправлений (≈4 часа)

Задача: Освоить технику внесения систематических исправлений.

1. Система отслеживания изменений:

python

```
class DocumentCorrector:
    """Управление процессом внесения исправлений"""

    def __init__(self, original_document):
        self.original_document = original_document
        self.corrections = []
        self.version_history = []

    def apply_correction(self, feedback_item, correction_text, correction_type):
        """Внесение исправления в документ"""
        correction = {
            'feedback_id': feedback_item['id'],
            'original_section': feedback_item['section'],
            'correction_text': correction_text,
            'correction_type': correction_type,
            'applied_by': 'Технический писатель',
            'applied_date': '2024-01-15',
            'status': 'completed',
            'before_after_comparison': None
        }

        self.corrections.append(correction)
        return correction

    def track_changes(self, section, original_text, corrected_text):
        """Отслеживание изменений в разделе"""
        change_record = {
            'section': section,
            'original': original_text,
            'corrected': corrected_text,
            'changes': self.calculate_changes(original_text, corrected_text),
            'timestamp': '2024-01-15 14:30:00'
        }

        self.version_history.append(change_record)
        return change_record

    def calculate_changes(self, original, corrected):
        """Вычисление внесенных изменений"""
        changes = {
            'characters_added': max(0, len(corrected) - len(original)),
            'characters_removed': max(0, len(original) - len(corrected)),
            'words_changed': self.count_word_changes(original, corrected),
            'sentences_modified': self.count_sentence_changes(original, corrected)
        }
        return changes

    def count_word_changes(self, original, corrected):
        """Подсчет измененных слов"""
        original_words = set(original.lower().split())
        corrected_words = set(corrected.lower().split())

        added_words = corrected_words - original_words
        removed_words = original_words - corrected_words

        return len(added_words) + len(removed_words)

    def count_sentence_changes(self, original, corrected):
        """Подсчет измененных предложений"""
        import re
        original_sentences = re.split(r'[.!?]+', original)
        corrected_sentences = re.split(r'[.!?]+', corrected)

        return abs(len(original_sentences) - len(corrected_sentences))

    def generate_correction_report(self):
        """Генерация отчета о внесенных исправлениях"""
        report = "# Отчет о внесенных исправлениях\n\n"
        report += f"**Всего исправлений:** {len(self.corrections)}\n"
        report += f"**Период:** 2024-01-15\n\n"

        report += "## Детали исправлений:\n\n"
        for correction in self.corrections:
            report += f"### Исправление #{correction['feedback_id']}\n"
```

```
            report += f"**Раздел:** {correction['original_section']}\n"
            report += f"**Тип исправления:** {correction['correction_type']}\n"
            report += f"**Исполнитель:** {correction['applied_by']}\n"
            report += f"**Статус:** {correction['status']}\n\n"

            if correction['before_after_comparison']:
                report += "**Сравнение:**\n"
                report += f"```diff\n{correction['before_after_comparison']}\n```\n\n"

        report += "## Статистика изменений:\n\n"
        total_changes = {
            'characters_added': 0,
            'characters_removed': 0,
            'words_changed': 0,
            'sections_modified': len(set(c['original_section'] for c in self.corrections))
        }

        for change in self.version_history:
            total_changes['characters_added'] += change['changes']['characters_added']
            total_changes['characters_removed'] += change['changes']['characters_removed']
            total_changes['words_changed'] += change['changes']['words_changed']

        for metric, value in total_changes.items():
            report += f"- **{metric.replace('_', ' ').title()}:** {value}\n"

        return report

# Пример использования
corrector = DocumentCorrector("Исходный документ")

# Пример исправления технической неточности
technical_feedback = {
    'id': 1,
    'section': 'Системные требования',
    'comment': 'Не указаны минимальные требования для Android 8.0'
}

corrector.apply_correction(
    technical_feedback,
    "**Минимальные системные требования:**\n- Android 8.0 (API 26) и выше\n- 2 ГБ оперативной памяти\n- 100 МБ
свободного места",
    "completeness_addition"
)

# Отслеживание изменений
original_text = "Приложение работает на Android."
corrected_text = "**Минимальные системные требования:**\n- Android 8.0 (API 26) и выше\n- 2 ГБ оперативной
памяти\n- 100 МБ свободного места"

corrector.track_changes("Системные требования", original_text, corrected_text)

report = corrector.generate_correction_report()
print(report)
```

2. Практическое задание "Исправление конкретных замечаний":

```
python
def apply_targeted_corrections(feedback_analyzer):
    """Применение целевых исправлений на основе анализа"""

    corrector = DocumentCorrector("Техническая документация")
    applied_corrections = []

    for feedback in feedback_analyzer.feedback_items:
        if feedback['priority'] == 'high':
            # Применение исправлений для высокоприоритетных замечаний
            if feedback['type'] == 'completeness':
                correction = apply_completeness_correction(feedback, corrector)
                applied_corrections.append(correction)
            elif feedback['type'] == 'technical_accuracy':
                correction = apply_technical_correction(feedback, corrector)
                applied_corrections.append(correction)

    return corrector, applied_corrections

def apply_completeness_correction(feedback, corrector):
    """Исправление недостающей информации"""
    if 'системные требования' in feedback['section'].lower():
        correction_text = """
```

Системные требования:

Минимальные требования:
- Android 8.0 (API 26) или выше
- 2 ГБ оперативной памяти
- 100 МБ свободного места на диске
- Интернет-соединение для синхронизации

Рекомендуемые требования:
- Android 10.0 (API 29) или выше
- 4 ГБ оперативной памяти
- 500 МБ свободного места
- Высокоскоростное интернет-соединение
"""
        return corrector.apply_correction(feedback, correction_text, "completeness_addition")

def apply_technical_correction(feedback, corrector):
    """Исправление технической неточности"""
    if 'скриншот' in feedback['comment'].lower():
        correction_text = """
Обновленные скриншоты интерфейса:

Все скриншоты в документации обновлены до версии 2.0 приложения.
Основные изменения интерфейса:
- Новая панель навигации
- Обновленная цветовая схема
- Улучшенная организация меню

![Обновленный интерфейс](images/updated_interface_v2.png)
"""
        return corrector.apply_correction(feedback, correction_text, "visual_update")

 # Применение исправлений
corrector, applied = apply_targeted_corrections(analyzer)
print(f"Применено исправлений: {len(applied)}")

Часть 4: Верификация и контроль качества (≈4 часа)
Задача: Освоить методы проверки качества внесенных исправлений.
1. Система верификации исправлений:
python
class QualityVerifier:
    """Верификация качества внесенных исправлений"""

    def __init__(self, original_analyzer, document_corrector):
        self.original_analyzer = original_analyzer
        self.corrector = document_corrector
        self.verification_results = []

    def verify_correction(self, correction):
        """Проверка отдельного исправления"""
        verification = {
            'correction_id': correction['feedback_id'],
            'section': correction['original_section'],
            'verification_status': 'pending',
            'verified_by': None,
            'verification_date': None,
            'issues_found': [],
            'quality_score': 0
        }

        # Проверка соответствия исходному замечанию
        original_feedback = next(
            (fb for fb in self.original_analyzer.feedback_items
             if fb['id'] == correction['feedback_id']),
            None
)

        if original_feedback:
            if self.check_resolution(original_feedback, correction):
                verification['verification_status'] = 'approved'
                verification['quality_score'] = self.calculate_quality_score(correction)
            else:
                verification['verification_status'] = 'needs_rework'
                verification['issues_found'].append('Исправление не полностью решает исходную проблему')

        verification['verified_by'] = 'Эксперт по качеству'

```
        verification['verification_date'] = '2024-01-16'

        self.verification_results.append(verification)
        return verification

    def check_resolution(self, original_feedback, correction):
        """Проверка решения исходной проблемы"""
        original_issue = original_feedback['comment'].lower()
        correction_text = correction['correction_text'].lower()

        # Простая проверка содержания
        if 'требован' in original_issue and 'требован' in correction_text:
            return True
        elif 'скриншот' in original_issue and 'скриншот' in correction_text:
            return True
        elif 'термин' in original_issue and 'термин' in correction_text:
            return True

        return False

    def calculate_quality_score(self, correction):
        """Расчет оценки качества исправления"""
        score = 50  # Базовый балл

        # Дополнительные баллы за качество
        if len(correction['correction_text']) > 100:
            score += 20  # Подробное исправление

        if '\n-' in correction['correction_text'] or '\n*' in correction['correction_text']:
            score += 15  # Структурированное исправление

        if '**' in correction['correction_text']:
            score += 10  # Использование форматирования

        if 'пример' in correction['correction_text'].lower():
            score += 5  # Включение примеров

        return min(score, 100)

    def generate_verification_report(self):
        """Генерация отчета верификации"""
        report = "# Отчет верификации исправлений\n\n"

        approved = [v for v in self.verification_results if v['verification_status'] == 'approved']
        needs_rework = [v for v in self.verification_results if v['verification_status'] == 'needs_rework']

        report += f"## Статистика верификации:\n\n"
        report += f"- **Всего проверено:** {len(self.verification_results)}\n"
        report += f"- **Утверждено:** {len(approved)}\n"
        report += f"- **Требуют доработки:** {len(needs_rework)}\n"
        report += f"- **Общий показатель качества:** {sum(v['quality_score'] for v in approved) / len(approved) if approved else
0:.1f}%\n\n"

        if needs_rework:
            report += "## Исправления, требующие доработки:\n\n"
            for verification in needs_rework:
                report += f"### {verification['section']}\n"
                report += f"**Проблемы:** {', '.join(verification['issues_found'])}\n\n"

        report += "## Детали проверки:\n\n"
        for verification in self.verification_results:
            status_icon = "  " if verification['verification_status'] == 'approved' else "⚠"
            report += f"{status_icon} **{verification['section']}** - Качество: {verification['quality_score']}%\n"

        return report

# Пример верификации
verifier = QualityVerifier(analyzer, corrector)

for correction in corrector.corrections:
    verifier.verify_correction(correction)

verification_report = verifier.generate_verification_report()
print(verification_report)
```

2. Практическое задание "Финальный контроль качества":

```
python
def perform_final_qa(original_doc, corrected_doc, feedback_analyzer):
```

```
    """Выполнение финального контроля качества"""

    qa_report = {
        'document': corrected_doc.document_name,
        'qa_date': '2024-01-16',
        'qa_engineer': 'Старший технический писатель',
        'checks_performed': [],
        'overall_status': 'pending'
    }

    # Проверка 1: Все ли замечания адресованы
    addressed_feedback = len(corrected_doc.corrections)
    total_feedback = len(feedback_analyzer.feedback_items)
    feedback_coverage = (addressed_feedback / total_feedback) * 100

    qa_report['checks_performed'].append({
        'check': 'Покрытие замечаний',
        'status': 'passed' if feedback_coverage >= 90 else 'failed',
        'details': f'Адресовано {addressed_feedback} из {total_feedback} замечаний ({feedback_coverage:.1f}%)'
    })

    # Проверка 2: Качество исправлений
    verifier = QualityVerifier(feedback_analyzer, corrected_doc)
    for correction in corrected_doc.corrections:
        verifier.verify_correction(correction)

    approved_corrections = [v for v in verifier.verification_results if v['verification_status'] == 'approved']
    correction_quality = (len(approved_corrections) / len(verifier.verification_results)) * 100

    qa_report['checks_performed'].append({
        'check': 'Качество исправлений',
        'status': 'passed' if correction_quality >= 80 else 'failed',
        'details': f'Утверждено {len(approved_corrections)} из {len(verifier.verification_results)} исправлений
({correction_quality:.1f}%)'
    })

    # Проверка 3: Соответствие стандартам
    standards_compliance = check_standards_compliance(corrected_doc)
    qa_report['checks_performed'].append({
        'check': 'Соответствие стандартам',
        'status': 'passed' if standards_compliance else 'failed',
        'details': 'Проверка форматирования, терминологии и структуры'
    })

    # Определение общего статуса
    passed_checks = sum(1 for check in qa_report['checks_performed'] if check['status'] == 'passed')
    total_checks = len(qa_report['checks_performed'])

    if passed_checks == total_checks:
        qa_report['overall_status'] = 'approved'
    elif passed_checks >= total_checks * 0.7:
        qa_report['overall_status'] = 'conditionally_approved'
    else:
        qa_report['overall_status'] = 'rejected'

    return qa_report

def check_standards_compliance(corrected_doc):
    """Проверка соответствия стандартам документации"""
    # Упрощенная проверка - в реальности была бы более сложная логика
    has_headers = any('#' in correction['correction_text'] for correction in corrected_doc.corrections)
    has_lists = any('-' in correction['correction_text'] or '*' in correction['correction_text'] for correction in
corrected_doc.corrections)
    has_formatting = any('**' in correction['correction_text'] for correction in corrected_doc.corrections)

    return has_headers and has_lists and has_formatting

# Выполнение финального QA
final_qa = perform_final_qa("Исходный документ", corrector, analyzer)

print("Финальный отчет контроля качества:")
for check in final_qa['checks_performed']:
    status_icon = "  " if check['status'] == 'passed' else "  "
    print(f"{status_icon} {check['check']}: {check['details']}")

print(f"\nОбщий статус: {final_qa['overall_status']}")
```

Итоговое задание для самостоятельного выполнения:
Сценарий: Полный цикл обработки экспертных замечаний по техническому документу.
Задание:
1. Исходные данные:
python
```
# Экспертные замечания к документу "API Руководство"
api_feedback = [
    {
        'expert': 'Senior Developer',
        'section': 'Аутентификация',
        'comment': 'Не описана процедура получения OAuth токена',
        'type': 'completeness',
        'priority': 'high'
    },
    {
        'expert': 'QA Lead',
        'section': 'Error Handling',
        'comment': 'Отсутствуют примеры обработки ошибок 429 (Too Many Requests)',
        'type': 'completeness',
        'priority': 'high'
    },
    {
        'expert': 'Technical Writer',
        'section': 'Введение',
        'comment': 'Слишком технический язык, нужно адаптировать для бизнес-пользователей',
        'type': 'clarity',
        'priority': 'medium'
    },
    {
        'expert': 'Product Manager',
        'section': 'Rate Limiting',
        'comment': 'Не указаны конкретные лимиты для разных endpoint',
        'type': 'technical_accuracy',
        'priority': 'high'
    }
]
```
2. Требуемые этапы:
Этап 1: Анализ и категоризация
- Систематизировать все замечания
- Определить приоритеты и категории
- Создать матрицу важности/срочности

Этап 2: Планирование исправлений
- Разработать план работ по спринтам
- Назначить ответственных
- Определить сроки выполнения

Этап 3: Внесение исправлений
- Реализовать все запланированные правки
- Документировать изменения
- Отслеживать прогресс

Этап 4: Верификация качества
- Проверить корректность исправлений
- Убедиться в решении исходных проблем
- Подготовить финальный отчет

3. Критерии оценки:

Компонент	Критерии
Системность анализа	Полнота категоризации, обоснованность приоритето
Эффективность планирования	Реалистичность сроков, оптимальное распределение
Качество исправлений	Полнота решения проблем, соответствие стандартам
Документирование процесса	Подробность отчетов, отслеживаемость изменений
Финальная верификация	Тщательность проверки, объективность оценки

4. Ожидаемые результаты:

1. **Отчет анализа замечаний** с распределением по категориям
2. **План исправлений** с сроками и ответственными
3. **Исправленный документ** с track changes
4. **Отчет верификации** с метриками качества
5. **Финальный QA отчет** с рекомендациями

Это занятие даст студентам комплексные навыки управления процессом ревизии документации - от анализа замечаний до финального контроля качества, что является критически важной компетенцией для технических писателей и документалистов.

Практическое занятие 87 - 90. Преобразование документов в различные выходные форматы

Отлично! Разработаем комплексный план практического занятия по преобразованию документов в различные форматы.

Практическое занятие 87-90: Преобразование документов в различные выходные форматы

Цель занятия: Освоить методы и инструменты преобразования технических документов между различными форматами (PDF, HTML, DOCX, EPUB, MOBI) с сохранением структуры и форматирования.

Программное обеспечение:

- **Конвертеры:** Pandoc, LibreOffice, Calibre
- **Языки разметки:** Markdown, reStructuredText, AsciiDoc
- **Программирование:** Python (pandoc-py, python-docx), Node.js
- **Веб-технологии:** HTML/CSS, PDF generators (WeasyPrint, PrinceXML)
- **Системы сборки:** Make, Gradle, GitHub Actions

Необходимые знания: Базовые навыки работы с текстовыми форматами, основы командной строки.

План занятия и Ход работы

Часть 1: Основы преобразования форматов (≈4 часа)

Задача: Понять принципы конвертации между основными форматами документов.

1. Матрица преобразования форматов:

python

```
class FormatConversionMatrix:
    """Матрица поддерживаемых преобразований между форматами"""

    def __init__(self):
        self.supported_conversions = {
            'markdown': {
                'to': ['html', 'pdf', 'docx', 'epub', 'latex', 'rst'],
                'tools': ['pandoc', 'python-markdown', 'marked'],
                'complexity': 'low'
            },
            'html': {
                'to': ['pdf', 'epub', 'mobi', 'docx', 'markdown'],
                'tools': ['pandoc', 'weasyprint', 'princexml'],
                'complexity': 'medium'
            },
            'docx': {
                'to': ['pdf', 'html', 'markdown', 'epub', 'odt'],
                'tools': ['pandoc', 'libreoffice', 'python-docx2txt'],
                'complexity': 'medium'
            },
            'pdf': {
                'to': ['html', 'text', 'docx'],
                'tools': ['pandoc', 'pdf2htmlEX', 'pdftotext'],
                'complexity': 'high'
            },
            'epub': {
                'to': ['mobi', 'pdf', 'html', 'markdown'],
                'tools': ['pandoc', 'calibre', 'ebook-convert'],
                'complexity': 'medium'
            }
        }

    def get_conversion_path(self, source_format, target_format):
        """Определение оптимального пути конвертации"""
        if source_format in self.supported_conversions:
            if target_format in self.supported_conversions[source_format]['to']:
                return {
                    'direct': True,
                    'path': [source_format, target_format],
                    'tools': self.supported_conversions[source_format]['tools'],
                    'complexity': self.supported_conversions[source_format]['complexity']
                }

        # Поиск непрямого пути через промежуточные форматы
        for intermediate in self.supported_conversions:
            if (source_format in self.supported_conversions and
                intermediate in self.supported_conversions[source_format]['to'] and
                target_format in self.supported_conversions[intermediate]['to']):
```

```
                return {
                    'direct': False,
                    'path': [source_format, intermediate, target_format],
                    'tools': list(set(
                        self.supported_conversions[source_format]['tools'] +
                        self.supported_conversions[intermediate]['tools']
                    )),
                    'complexity': 'high'
                }

        return None

    def generate_conversion_guide(self, source_format, target_format):
        """Генерация руководства по конвертации"""
        conversion = self.get_conversion_path(source_format, target_format)

        if not conversion:
            return f"Конвертация из {source_format} в {target_format} не поддерживается"

        guide = f"# Руководство по конвертации: {source_format.upper()} → {target_format.upper()}\n\n"
        guide += f"**Путь конвертации:** {' → '.join(conversion['path'])}\n"
        guide += f"**Сложность:** {conversion['complexity']}\n"
        guide += f"**Рекомендуемые инструменты:** {', '.join(conversion['tools'][:3])}\n\n"

        guide += "## Рекомендуемые команды:\n\n"

        if 'pandoc' in conversion['tools']:
            guide += "### Использование Pandoc:\n"
            guide += f"```bash\npandoc input.{source_format} -o output.{target_format}\n```\n\n"

        if 'libreoffice' in conversion['tools'] and source_format == 'docx':
            guide += "### Использование LibreOffice:\n"
            guide += f"```bash\nlibreoffice --headless --convert-to {target_format} input.{source_format}\n```\n"

        return guide

# Пример использования
matrix = FormatConversionMatrix()

# Прямая конвертация
markdown_to_html = matrix.get_conversion_path('markdown', 'html')
print("Конвертация Markdown → HTML:")
print(f"Путь: {' → '.join(markdown_to_html['path'])}")
print(f"Инструменты: {', '.join(markdown_to_html['tools'])}")

# Непрямая конвертация
docx_to_epub = matrix.get_conversion_path('docx', 'epub')
print(f"\nКонвертация DOCX → EPUB:")
print(f"Путь: {' → '.join(docx_to_epub['path'])}")

# Генерация руководства
guide = matrix.generate_conversion_guide('markdown', 'pdf')
print(f"\n{guide}")
```

2. Практическое задание "Анализ требований к конвертации":

```
python
def analyze_document_conversion_needs(document_spec):
    """Анализ требований к конвертации документа"""

    analysis = {
        'document': document_spec['name'],
        'source_format': document_spec['source_format'],
        'target_formats': [],
        'special_requirements': [],
        'conversion_plan': []
    }

    # Определение целевых форматов на основе использования
    use_cases = document_spec.get('use_cases', [])

    for use_case in use_cases:
        if use_case == 'web_publishing':
            analysis['target_formats'].append('html')
            analysis['special_requirements'].append('responsive_design')
        elif use_case == 'print':
            analysis['target_formats'].append('pdf')
            analysis['special_requirements'].append('print_optimization')
        elif use_case == 'ebook':
```

```
            analysis['target_formats'].append('epub')
            analysis['target_formats'].append('mobi')
            analysis['special_requirements'].append('reflowable_layout')
        elif use_case == 'office_use':
            analysis['target_formats'].append('docx')
            analysis['special_requirements'].append('editable_format')

    # Создание плана конвертации
    matrix = FormatConversionMatrix()
    for target_format in analysis['target_formats']:
        conversion_path = matrix.get_conversion_path(
            document_spec['source_format'],
            target_format
        )
        analysis['conversion_plan'].append({
            'target': target_format,
            'path': conversion_path['path'],
            'tools': conversion_path['tools'],
            'complexity': conversion_path['complexity']
        })

    return analysis

# Пример документа
document_spec = {
    'name': 'Technical User Manual',
    'source_format': 'markdown',
    'use_cases': ['web_publishing', 'print', 'ebook', 'office_use']
}

conversion_analysis = analyze_document_conversion_needs(document_spec)
print("Анализ требований к конвертации:")
print(f"Документ: {conversion_analysis['document']}")
print(f"Исходный формат: {conversion_analysis['source_format']}")
print(f"Целевые форматы: {', '.join(conversion_analysis['target_formats'])}")
print(f"Особые требования: {', '.join(conversion_analysis['special_requirements'])}")
```

Часть 2: Конвертация с использованием Pandoc (≈4 часа)
Задача: Освоить мощный инструмент конвертации Pandoc.
1. Python обертка для Pandoc:
python

```
import subprocess
import os
import json
from pathlib import Path

class PandocConverter:
    """Управление конвертацией документов через Pandoc"""

    def __init__(self, pandoc_path='pandoc'):
        self.pandoc_path = pandoc_path
        self.supported_formats = self.get_supported_formats()

    def get_supported_formats(self):
        """Получение списка поддерживаемых форматов"""
        try:
            result = subprocess.run(
                [self.pandoc_path, '--list-input-formats'],
                capture_output=True, text=True, check=True
            )
            input_formats = result.stdout.strip().split('\n')

            result = subprocess.run(
                [self.pandoc_path, '--list-output-formats'],
                capture_output=True, text=True, check=True
            )
            output_formats = result.stdout.strip().split('\n')

            return {
                'input': input_formats,
                'output': output_formats
            }
        except (subprocess.CalledProcessError, FileNotFoundError):
            return {'input': [], 'output': []}

    def convert_document(self, input_file, output_format, output_file=None, options=None):
```

```
    """Конвертация документа в целевой формат"""

    if output_file is None:
        input_path = Path(input_file)
        output_file = input_path.with_suffix(f'.{output_format}')

    # Базовые параметры pandoc
    cmd = [self.pandoc_path, input_file, '-o', output_file]

    # Добавление опций
    if options:
        if 'template' in options:
            cmd.extend(['--template', options['template']])
        if 'toc' in options and options['toc']:
            cmd.append('--toc')
        if 'number-sections' in options and options['number-sections']:
            cmd.append('--number-sections')
        if 'css' in options:
            cmd.extend(['--css', options['css']])
        if 'metadata' in options:
            for key, value in options['metadata'].items():
                cmd.extend(['-M', f'{key}={value}'])

    try:
        result = subprocess.run(cmd, capture_output=True, text=True, check=True)

        return {
            'success': True,
            'output_file': str(output_file),
            'input_size': os.path.getsize(input_file),
            'output_size': os.path.getsize(output_file),
            'command': ' '.join(cmd)
        }

    except subprocess.CalledProcessError as e:
        return {
            'success': False,
            'error': e.stderr,
            'command': ' '.join(cmd)
        }

def batch_convert(self, input_files, output_format, output_dir, options=None):
    """Пакетная конвертация нескольких файлов"""

    results = []
    output_dir = Path(output_dir)
    output_dir.mkdir(exist_ok=True)

    for input_file in input_files:
        input_path = Path(input_file)
        output_file = output_dir / input_path.with_suffix(f'.{output_format}').name

        result = self.convert_document(
            input_file,
            output_format,
            output_file,
            options
        )

        results.append({
            'input_file': input_file,
            'output_file': str(output_file),
            'success': result['success']
        })

    return results

def create_conversion_preset(self, preset_name, options):
    """Создание пресетов конвертации для различных целей"""

    presets = {
        'web_optimized': {
            'options': {
                'toc': True,
                'number-sections': True,
                'css': 'styles/web.css',
                'metadata': {
```

```
                'title': 'Document',
                'author': 'Technical Writer'
            }
        },
        'description': 'Оптимизировано для веб-публикации'
    },
    'print_ready': {
        'options': {
            'toc': True,
            'number-sections': True,
            'css': 'styles/print.css',
            'metadata': {
                'papersize': 'a4',
                'class': 'article'
            }
        },
        'description': 'Оптимизировано для печати'
    },
    'ebook': {
        'options': {
            'toc': True,
            'css': 'styles/ebook.css',
            'metadata': {
                'title': 'Document',
                'language': 'ru'
            }
        },
        'description': 'Оптимизировано для электронных книг'
    }
}

presets[preset_name] = {
    'options': options,
    'description': 'Пользовательский пресет'
}

return presets[preset_name]

# Пример использования
converter = PandocConverter()

# Проверка доступности pandoc
if converter.supported_formats['input']:
    print("Pandoc доступен. Поддерживаемые форматы:")
    print(f"Входные: {', '.join(converter.supported_formats['input'][:5])}...")
    print(f"Выходные: {', '.join(converter.supported_formats['output'][:5])}...")
else:
    print("Pandoc не найден. Установите pandoc для работы с конвертацией.")

# Создание пресетов
web_preset = converter.create_conversion_preset('web_optimized', {
    'toc': True,
    'css': 'styles.css',
    'metadata': {'title': 'My Document'}
})

print(f"\nПресет 'web_optimized': {web_preset['description']}")
```

2. Практическое задание "Конвертация технического руководства":

```
python
def convert_technical_manual():
    """Конвертация технического руководства в различные форматы"""

    converter = PandocConverter()

    # Предполагаем, что у нас есть markdown файлы
    input_files = [
        'docs/introduction.md',
        'docs/installation.md',
        'docs/usage.md',
        'docs/api.md',
        'docs/troubleshooting.md'
    ]

    # Создание различных версий
    conversion_jobs = [
        {
            'format': 'html',
```

```
            'preset': 'web_optimized',
            'output_dir': 'build/html',
            'description': 'Веб-версия документации'
        },
        {
            'format': 'pdf',
            'preset': 'print_ready',
            'output_dir': 'build/pdf',
            'description': 'Версия для печати'
        },
        {
            'format': 'epub',
            'preset': 'ebook',
            'output_dir': 'build/ebook',
            'description': 'Электронная книга'
        },
        {
            'format': 'docx',
            'preset': None,
            'output_dir': 'build/word',
            'description': 'Версия для редактирования'
        }
    ]

    results = {}

    for job in conversion_jobs:
        print(f"Конвертация в {job['format'].upper()}...")

        options = None
        if job['preset']:
            preset = converter.create_conversion_preset(job['preset'], {})
            options = preset['options']

        result = converter.batch_convert(input_files, job['format'], job['output_dir'], options)
        results[job['format']] = {
            'description': job['description'],
            'files': result,
            'success_count': sum(1 for r in result if r['success'])
        }

    # Генерация отчета
    report = "# Отчет конвертации технического руководства\n\n"

    for format, data in results.items():
        status = "    Успешно" if data['success_count'] == len(input_files) else "⚠ С ошибками"
        report += f"## {format.upper()} - {data['description']}\n"
        report += f"**Статус:** {status}\n"
        report += f"**Успешно сконвертировано:** {data['success_count']}/{len(input_files)}\n\n"

    return report

# Запуск конвертации
conversion_report = convert_technical_manual()
print(conversion_report)
```

Часть 3: Продвинутые техники конвертации (≈4 часа)
Задача: Освоить обработку сложных элементов и кастомизацию вывода.
1. Обработка сложных структур:
python
```
class AdvancedDocumentConverter:
    """Продвинутая конвертация с обработкой сложных элементов"""

    def __init__(self, converter):
        self.converter = converter
        self.preprocessors = []
        self.postprocessors = []

    def add_preprocessor(self, name, function):
        """Добавление препроцессора для обработки контента"""
        self.preprocessors.append({
            'name': name,
            'function': function
        })

    def add_postprocessor(self, name, function):
```

```
        """Добавление постпроцессора для обработки вывода"""
        self.postprocessors.append({
            'name': name,
            'function': function
        })

    def preprocess_content(self, content, source_format):
        """Предварительная обработка контента"""
        for processor in self.preprocessors:
            content = processor['function'](content, source_format)
        return content

    def postprocess_content(self, content, target_format):
        """Постобработка контента"""
        for processor in self.postprocessors:
            content = processor['function'](content, target_format)
        return content

    def convert_with_processing(self, input_file, output_format, output_file, options=None):
        """Конвертация с полной обработкой контента"""

        # Чтение исходного файла
        with open(input_file, 'r', encoding='utf-8') as f:
            content = f.read()

        # Предварительная обработка
        processed_content = self.preprocess_content(content, Path(input_file).suffix[1:])

        # Создание временного файла с обработанным контентом
        temp_file = Path(input_file).with_suffix('.processed' + Path(input_file).suffix)
        with open(temp_file, 'w', encoding='utf-8') as f:
            f.write(processed_content)

        # Конвертация
        result = self.converter.convert_document(
            str(temp_file),
            output_format,
            output_file,
            options
        )

        # Постобработка если конвертация успешна
        if result['success'] and Path(output_file).exists():
            with open(output_file, 'r', encoding='utf-8') as f:
                output_content = f.read()

            final_content = self.postprocess_content(output_content, output_format)

            with open(output_file, 'w', encoding='utf-8') as f:
                f.write(final_content)

        # Удаление временного файла
        if temp_file.exists():
            temp_file.unlink()

        return result

# Примеры препроцессоров и постпроцессоров
def diagram_preprocessor(content, source_format):
    """Препроцессор для обработки диаграмм Mermaid"""
    import re

    # Замена mermaid блоков на изображения
    pattern = r'```mermaid\s*(.*?)\s*```'

    def replace_mermaid(match):
        mermaid_code = match.group(1)
        # В реальной реализации здесь был бы вызов API mermaid для генерации изображения
        return f'![Диаграмма](generated/diagram_{hash(mermaid_code)}.png)'

    return re.sub(pattern, replace_mermaid, content, flags=re.DOTALL)

def table_formatter_postprocessor(content, target_format):
    """Постпроцессор для форматирования таблиц"""
    if target_format == 'html':
        # Добавление CSS классов для таблиц
        content = content.replace('<table>', '<table class="table table-striped">')
```

```
    return content

def code_syntax_highlighter(content, target_format):
    """Постпроцессор для подсветки синтаксиса"""
    if target_format == 'html':
        # Добавление классов для highlight.js
        import re
        pattern = r'<code class="language-(\w+)">'
        replacement = r'<code class="language-\1 hljs">'
        content = re.sub(pattern, replacement, content)

    return content

# Использование продвинутого конвертера
advanced_converter = AdvancedDocumentConverter(PandocConverter())

# Регистрация процессоров
advanced_converter.add_preprocessor('mermaid_diagrams', diagram_preprocessor)
advanced_converter.add_postprocessor('table_formatting', table_formatter_postprocessor)
advanced_converter.add_postprocessor('syntax_highlighting', code_syntax_highlighter)

print("Продвинутый конвертер инициализирован с процессорами:")
for pre in advanced_converter.preprocessors:
    print(f" - Препроцессор: {pre['name']}")
for post in advanced_converter.postprocessors:
    print(f" - Постпроцессор: {post['name']}")
```

2. Практическое задание "Кастомизация вывода":

```
python
def create_custom_templates():
    """Создание кастомных шаблонов для различных форматов"""

    templates = {
        'html': {
            'template': """
<!DOCTYPE html>
<html lang="ru">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>$title$</title>
    <style>
        body { font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif; line-height: 1.6; margin: 0; padding: 20px; }
        .container { max-width: 800px; margin: 0 auto; }
        h1 { color: #2c3e50; border-bottom: 2px solid #3498db; padding-bottom: 10px; }
        code { background: #f8f9fa; padding: 2px 4px; border-radius: 3px; }
        pre { background: #2c3e50; color: #ecf0f1; padding: 15px; border-radius: 5px; overflow-x: auto; }
        table { width: 100%; border-collapse: collapse; margin: 20px 0; }
        th, td { padding: 12px; text-align: left; border-bottom: 1px solid #ddd; }
        th { background-color: #3498db; color: white; }
    </style>
</head>
<body>
    <div class="container">
        $body$
    </div>
</body>
</html>
            """,
            'description': 'Кастомный HTML шаблон с улучшенным дизайном'
        },

        'latex': {
            'template': """
\\documentclass[12pt,a4paper]{article}
\\usepackage[utf8]{inputenc}
\\usepackage[russian]{babel}
\\usepackage{geometry}
\\geometry{left=2cm,right=2cm,top=2cm,bottom=2cm}

\\title{$title$}
\\author{$author$}
\\date{\\today}

\\begin{document}

\\maketitle
```

```
\\tableofcontents
\\newpage

$body$

\\end{document}
        """,
        'description': 'LaTeX шаблон для академических документов'
    },

    'epub': {
        'template': """
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:epub="http://www.idpf.org/2007/ops">
<head>
  <title>$title$</title>
  <style>
    body { font-family: serif; line-height: 1.6; }
    h1 { text-align: center; }
    .page-break { page-break-before: always; }
  </style>
</head>
<body>
  $body$
</body>
</html>
        """,
        'description': 'EPUB шаблон с поддержкой электронных книг'
    }
}

# Сохранение шаблонов в файлы
for format, template_data in templates.items():
    filename = f"templates/custom.{format}"
    Path('templates').mkdir(exist_ok=True)

    with open(filename, 'w', encoding='utf-8') as f:
        f.write(template_data['template'])

    print(f"Создан шаблон: {filename} - {template_data['description']}")

return templates

# Создание кастомных шаблонов
custom_templates = create_custom_templates()
```

Часть 4: Автоматизация и пайплайны (≈4 часа)
Задача: Создать автоматизированные пайплайны для конвертации документов.
1. Система автоматической сборки:

```
python
import yaml
from datetime import datetime
import hashlib

class DocumentPipeline:
    """Пайплайн для автоматической конвертации документов"""

    def __init__(self, config_file='pipeline.yaml'):
        self.config = self.load_config(config_file)
        self.converter = AdvancedDocumentConverter(PandocConverter())
        self.build_cache = {}

    def load_config(self, config_file):
        """Загрузка конфигурации пайплайна"""
        default_config = {
            'source': {
                'format': 'markdown',
                'directory': 'src',
                'patterns': ['*.md', '*.markdown']
            },
            'build': {
                'directory': 'build',
                'clean_before_build': True,
                'cache_enabled': True
            },
```

```
        'targets': [
            {
                'name': 'web',
                'format': 'html',
                'output_dir': 'build/web',
                'options': {
                    'toc': True,
                    'css': 'assets/web.css',
                    'template': 'templates/custom.html'
                }
            },
            {
                'name': 'pdf',
                'format': 'pdf',
                'output_dir': 'build/pdf',
                'options': {
                    'toc': True,
                    'number-sections': True,
                    'template': 'templates/custom.latex'
                }
            },
            {
                'name': 'ebook',
                'format': 'epub',
                'output_dir': 'build/ebook',
                'options': {
                    'toc': True,
                    'template': 'templates/custom.epub'
                }
            }
        ]
    }

    if Path(config_file).exists():
        with open(config_file, 'r', encoding='utf-8') as f:
            user_config = yaml.safe_load(f)
            # Объединение конфигураций
            default_config.update(user_config)

    return default_config

def get_source_files(self):
    """Получение списка исходных файлов"""
    source_dir = Path(self.config['source']['directory'])
    patterns = self.config['source']['patterns']

    files = []
    for pattern in patterns:
        files.extend(source_dir.glob(pattern))

    return sorted(files)

def calculate_file_hash(self, file_path):
    """Вычисление хеша файла для кэширования"""
    with open(file_path, 'rb') as f:
        return hashlib.md5(f.read()).hexdigest()

def needs_rebuild(self, source_file, target_file):
    """Проверка необходимости пересборки"""
    if not self.config['build']['cache_enabled']:
        return True

    if not target_file.exists():
        return True

    source_hash = self.calculate_file_hash(source_file)
    target_mtime = target_file.stat().st_mtime

    # Проверка кэша
    cache_key = f"{source_file}_{target_file}"
    if cache_key in self.build_cache:
        cached_hash, cached_time = self.build_cache[cache_key]
        if cached_hash == source_hash and cached_time >= target_mtime:
            return False

    return True
```

```
def build_target(self, target_config):
    """Сборка конкретной цели"""
    print(f"Сборка цели: {target_config['name']}")

    source_files = self.get_source_files()
    output_dir = Path(target_config['output_dir'])
    output_dir.mkdir(parents=True, exist_ok=True)

    results = []

    for source_file in source_files:
        output_file = output_dir / source_file.with_suffix(f'.{target_config["format"]}').name

        if self.needs_rebuild(source_file, output_file):
            print(f"  Конвертация: {source_file.name} → {output_file.name}")

            result = self.converter.convert_with_processing(
                str(source_file),
                target_config['format'],
                str(output_file),
                target_config.get('options', {})
            )

            if result['success']:
                # Обновление кэша
                source_hash = self.calculate_file_hash(source_file)
                self.build_cache[f"{source_file}_{output_file}"] = (
                    source_hash,
                    datetime.now().timestamp()
                )

            results.append({
                'source': source_file.name,
                'target': output_file.name,
                'success': result['success'],
                'size': output_file.stat().st_size if output_file.exists() else 0
            })
        else:
            print(f"  Пропуск (актуально): {source_file.name}")
            results.append({
                'source': source_file.name,
                'target': output_file.name,
                'success': True,
                'cached': True,
                'size': output_file.stat().st_size
            })

    return results

def run_pipeline(self):
    """Запуск полного пайплайна"""
    print("Запуск пайплайна конвертации документов...")

    # Очистка build директории если нужно
    if self.config['build']['clean_before_build']:
        build_dir = Path(self.config['build']['directory'])
        if build_dir.exists():
            import shutil
            shutil.rmtree(build_dir)

    # Сборка всех целей
    pipeline_results = {}

    for target in self.config['targets']:
        results = self.build_target(target)
        pipeline_results[target['name']] = {
            'format': target['format'],
            'results': results,
            'success_count': sum(1 for r in results if r['success']),
            'total_count': len(results)
        }

    return self.generate_build_report(pipeline_results)

def generate_build_report(self, results):
    """Генерация отчета о сборке"""
    report = f"# Отчет сборки документов\n\n"
```

```
        report += f"**Время сборки:** {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}\n"
        report += f"**Всего целей:** {len(results)}\n\n"

        for target_name, target_data in results.items():
            status = "    Успешно" if target_data['success_count'] == target_data['total_count'] else "⚠ С ошибками"
            report += f"## Цель: {target_name} ({target_data['format'].upper()})\n"
            report += f"**Статус:** {status}\n"
            report += f"**Успешно:** {target_data['success_count']}/{target_data['total_count']}\n\n"

            for result in target_data['results']:
                icon = "    " if result['success'] else "    "
                cached = " (кэш)" if result.get('cached') else ""
                report += f"{icon} {result['source']} → {result['target']}{cached}\n"

            report += "\n"

        return report

# Пример конфигурационного файла pipeline.yaml
pipeline_config = """
source:
  format: markdown
  directory: "src"
  patterns: ["*.md", "*.markdown"]

build:
  directory: "build"
  clean_before_build: true
  cache_enabled: true

targets:
  - name: "web"
    format: "html"
    output_dir: "build/web"
    options:
      toc: true
      css: "assets/web.css"

  - name: "pdf"
    format: "pdf"
    output_dir: "build/pdf"
    options:
      toc: true
      number-sections: true

  - name: "ebook"
    format: "epub"
    output_dir: "build/ebook"
"""

# Сохранение конфигурации
with open('pipeline.yaml', 'w', encoding='utf-8') as f:
    f.write(pipeline_config)

# Запуск пайплайна
pipeline = DocumentPipeline('pipeline.yaml')
build_report = pipeline.run_pipeline()
print(build_report)
```

2. Практическое задание "CI/CD для документации":

```
python
def create_github_actions_workflow():
    """Создание GitHub Actions workflow для автоматической сборки"""

    workflow = """
name: Build Documentation

on:
  push:
    branches: [ main, master ]
  pull_request:
    branches: [ main, master ]

jobs:
  build:
    runs-on: ubuntu-latest

    steps:
```

```
    - uses: actions/checkout@v3

    - name: Setup Python
      uses: actions/setup-python@v4
      with:
        python-version: '3.9'

    - name: Install Pandoc
      run: |
        sudo apt-get update
        sudo apt-get install -y pandoc

    - name: Install dependencies
      run: |
        pip install -r requirements.txt

    - name: Build documentation
      run: |
        python build_pipeline.py

    - name: Upload PDF artifacts
      uses: actions/upload-artifact@v3
      with:
        name: pdf-documentation
        path: build/pdf/

    - name: Upload HTML artifacts
      uses: actions/upload-artifact@v3
      with:
        name: html-documentation
        path: build/html/

    - name: Deploy to GitHub Pages
      if: github.ref == 'refs/heads/main'
      uses: peaceiris/actions-gh-pages@v3
      with:
        github_token: ${{ secrets.GITHUB_TOKEN }}
        publish_dir: ./build/html
"""

    Path('.github/workflows').mkdir(parents=True, exist_ok=True)
    with open('.github/workflows/docs.yml', 'w', encoding='utf-8') as f:
        f.write(workflow)

    print("Создан GitHub Actions workflow: .github/workflows/docs.yml")

# Создание workflow
create_github_actions_workflow()
```

Итоговое задание для самостоятельного выполнения:
Сценарий: Создание системы конвертации технической документации проекта.
Задание:
1. Структура проекта:
text

```
documentation-project/
├── src/
│   ├── introduction.md
│   ├── installation.md
│   ├── usage.md
│   ├── api-reference.md
│   └── troubleshooting.md
├── templates/
│   ├── custom.html
│   ├── custom.latex
│   └── custom.epub
├── assets/
│   ├── web.css
│   └── print
```