МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение высшего образования
«КУЗБАССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ Т. Ф. ГОРБАЧЕВА»
Филиал КузГТУ в г. Белово

Кафедра инженерно-экономическая

ПМ.02 Осуществление интеграции программных модулей
МДК.02.01 Технология разработки программного обеспечения

Методические рекомендации
по выполнению самостоятельных работ
для специальности
09.02.07 «Информационные системы и программирование»

Составитель: Витвицкий М.Н.
Рассмотрены и утверждены на заседании кафедры
Протокол № 6 от 14.02.2026 г.
Рекомендовано учебно-методической комиссией специальностей СПО в качестве электронного издания для использования в учебном процессе
Протокол № 6 от 17.02.2026 г.

Долганова Жанна Александровна
Подписано цифровой подписью: Долганова Жанна Александровна
Дата: 2026.02.18 11:16:19 +07'00'

Белово 2026

СОДЕРЖАНИЕ

ОРГАНИЗАЦИЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

Самостоятельная работа обучающихся может рассматриваться как организационная форма обучения, обеспечивающих управление учебной деятельностью или деятельность обучающихся по освоению общих и профессиональных компетенций, знаний и умений учебной и научной деятельности без посторонней помощи.

В учебном процессе выделяют два вида самостоятельной работы: аудиторная, внеаудиторная.

Аудиторная самостоятельная работа по учебной дисциплине и профессиональному модулю выполняется на учебных занятиях под непосредственным руководством преподавателя и по его заданию.

Внеаудиторная самостоятельная работа выполняется учащимся по заданию преподавателя, но без его непосредственного участия.

Самостоятельная работа обучающихся проводится с целью: систематизации и закрепления полученных теоретических знании и

практических умений студентов;

углубления и расширения теоретических знаний;

формирования умений использовать нормативную, правовую, справочную документацию и специальную литературу;

развития познавательных способностей и активности обучающихся: творческой инициативы, самостоятельности, ответственности и организованности;

формирования самостоятельности мышления, способностей к саморазвитию, самосовершенствованию и самореализации;

развития исследовательских умений;

формирования общих и профессиональных компетенций.

ПЛАНИРОВАНИЕ ВНЕАУДИТОРНОЙ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

Преподавателем учебной дисциплины эмпирически определяются затраты времени на самостоятельное выполнение конкретного содержания учебного задания: на основании наблюдений за выполнением учащимися аудиторной самостоятельной работы, опроса студентов о затратах времени на то или иное задание, хронометража собственных затрат на решение той или иной задачи с внесением поправочного коэффициента из расчета уровня знаний и умений обучающихся.

При разработке рабочей программы по учебной дисциплине или профессиональному модулю при планировании содержания внеаудиторной самостоятельной работы преподавателей устанавливается содержание и объем теоретической учебной информации или практических заданий, которые выносятся на внеаудиторную самостоятельную работу, определяются формы и методы контроля результатов.

Содержание внеаудиторной самостоятельной работы определяется в соответствии с рекомендуемыми видами заданий согласно программе учебной дисциплины профессионального модуля.

Видами заданий для внеаудиторной самостоятельной работы могут быть:

•*для овладения знаниями:* компетентностно-ориентированные задание, чтение текста (учебника, первоисточника, дополнительной литературы): составление плана текста; графическое изображение структуры текста; конспектирование текста; реферирование текста; выписки из текста; работа со словарями и справочниками, ознакомление с нормативными документами; учебно-исследовательская работа; использование аудио- и видеозаписей, компьютерной техники и Интернета и др.;

•*для закрепления и систематизации знаний:* компетентностно-ориентированное задание, работа с конспектом лекции (обработка текста); повторная работа над учебным материалом (учебника, первоисточника, дополнительной литературы, аудио- и видеозаписей); составление плана и тезисов ответа; составление таблиц для систематизации учебного материала;

изучение нормативных материалов; ответы на контрольные вопросы; аналитическая обработка текста (аннотирование, рецензирование, реферирование, контент-анализ и др.); подготовка сообщений к выступлению на семинаре, конференции; подготовка рефератов, докладов; составление библиографии, тематических кроссвордов; тестирование и др.;

•*для формирования компетенций:* компетентностно-ориентированное задание, решение задач и упражнений по образцу; решение вариативных задач и упражнений; выполнение чертежей, схем; выполнение расчетно- графических работ; решение ситуационных педагогических задач; подготовка к деловым играм; проектирование и моделирование разных видов и компонентов профессиональной деятельности; подготовка курсовых работ; опытно-экспериментальная работа; упражнения на тренажере; упражнения спортивно-оздоровительного характера; рефлексивный анализ профессиональных умений с использованием аудио- и видеотехники и др.

Виды заданий для внеаудиторной самостоятельной работы, их содержание и характер могут иметь вариативный и дифференцированный характер, учитывать специфику специальности, изучаемой дисциплины, индивидуальные особенности студента.

При предъявлении видов заданий на внеаудиторную самостоятельную работу рекомендуется использовать дифференцированный подход к студентам. Перед выполнением студентами внеаудиторной самостоятельной работы преподаватель проводит инструктаж по выполнению задания, который включает цель задания, его содержание, сроки выполнения, ориентировочный объем работы, основные требования к результатам работы, критерии оценки. В процессе инструктажа преподаватель предупреждает обучающихся о возможных типичных ошибках, встречающихся при выполнении задания. Инструктаж проводится преподавателем за счет объема времени, отведенного на изучение дисциплины.

Самостоятельная работа может осуществляться индивидуально или группами обучающихся в зависимости от цели, объема, конкретной тематики самостоятельной работы, уровня сложности уровня умений обучающихся.

Отчет по самостоятельной работе обучающихся предоставляется в электронном виде.

КОНТРОЛЬ РЕЗУЛЬТАТОВ ВНЕАУДИТОРНОЙ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

Контроль результатов внеаудиторной самостоятельной работы студентов может осуществляться в пределах времени, отведенного на обязательные учебные занятия по дисциплине и внеаудиторную самостоятельную работу обучающихся по дисциплине, может проходить в письменной, устной или смешанной форме, с представлением продукта деятельности учащегося.

В качестве форм и методов контроля внеаудиторной самостоятельной работы обучающихся могут быть использованы*, зачеты, тестирование, самоотчеты, контрольные работы, защита творческих работ и др., которые могут осуществляться на учебном занятии или вне его (например, оценки за реферат).*

Критериями оценки результатов внеаудиторной самостоятельной работы обучающегося являются:

уровень освоения учащимся учебного материала;

умение обучающегося использовать теоретические знания при выполнении практических задач;

сформированность общих и профессиональных компетенций;

обоснованность и четкость изложения ответа;

оформление материала в соответствии с требованиями.

МЕТОДИЧЕСКИЕ МАТЕРИАЛЫ

РЕФЕРАТ

Реферат (от латинского – сообщаю) – краткое изложение в письменном виде содержания научного труда (трудов), литературы по теме. Это самостоятельная научно-исследовательская работа, где раскрывается суть исследуемой проблемы, изложение материала носит проблемно-тематический характер, показываются различные точки зрения, а также собственные взгляды на проблему. Содержание реферата должно быть логичным.

Критерии оценки реферата:

•соответствие теме;

•глубина проработки материала:

•правильность и полнота использования источников;

•оформление реферата.

ДОКЛАД

Доклад – вид самостоятельной работы обучающихся, используется в учебных и внеклассных занятиях, способствует формированию навыков исследовательской работы, расширяет познавательные интересы, приучает практически мыслить. При написании доклада по заданной теме следует составить план, подобрать основные источники. Работая с источниками, попытаться систематизировать полученные сведения, сделать выводы и обобщения. В настоящее время в учебных заведениях доклады содержательно практически ничем не отличаются от рефератов. Структура и оформление доклада такое же, как в реферате.

Критерии оценки доклада:

•соответствие теме;

•глубина проработки материала:

•правильность и полнота использования источников;

•оформление доклада.

Самостоятельная работа состоит из 2 заданий:

1. Теоретическое задание (реферат, доклад на 10 стр. А4);
2. Практическое задание (выдается преподавателем индивидуально согласно перечню).

Оформление работы

На титульном листе посередине его записывается вид работы, ниже на 10 мм – её название строчными буквами, справа в нижнем углу – фамилия автора разработки, группа. В нижней части титульного листа посредине указывается год написания разработки.

При наборе рекомендуется использовать основные системные гарнитуры шрифта TimesNewRoman. Текст набирается с соблюдением следующих правил: не допускаются ручной набор нумерации в главах и абзацах (только автонумарация); два и более пробела между символами. При наборе должны различаться тире и дефисы; маркеры и другие знаки должны быть сохранены аналогичными на протяжении всего материала. Между инициалами и после них (перед фамилией) ставится неразрывный пробел.

Размеры полей «обычное»: верхнее 1 см, левое 2 см, нижнее 1 см, правое 1 см. Нумерация страниц – внизу «по центру» шрифтом 12 пт. гарнитуры шрифта TimesNewRoman, нумерация страниц записки сквозная, причем начинается простановка номеров со страницы «Содержание», с учетом всех впереди стоящих страниц, на которых номера не проставляются.

Темы самостоятельной работы

№ раздела (темы)	Вопросы, выносимые на самостоятельное изучение	Количество часов
ТЕМА 2.1.1. ОСНОВНЫЕ ПОНЯТИЯ И СТАНДАРТИЗАЦИЯ ТРЕБОВАНИЙ К ПРОГРАММНОМУ ОБЕСПЕЧЕНИЮ.	Самостоятельная работа №1. Анализ предметной области.	2
	Самостоятельная работа обучающихся №2 Разработка и оформление технического задания.	2
	Самостоятельная работа обучающихся №3. Изучение работы в системе контроля версий.	2
ТЕМА 2.1.2. ОПИСАНИЕ И АНАЛИЗ ТРЕБОВАНИЙ. ДИАГРАММЫ IDEF, МЕТОДОЛОГИЯ UML.	Самостоятельная работа обучающихся №4. Построение диаграммы Вариантов использования и диаграммы последовательности.	2

	Самостоятельная работа обучающихся №5. Построение диаграммы Кооперации и диаграммы развертывания.	2
	Самостоятельная работа обучающихся №6. Построение диаграммы Деятельности, диаграммы состояний и диаграммы классов.	2
	Самостоятельная работа обучающихся №7. Построение диаграмм потоков данных	2
ТЕМА 2.1.3. ОЦЕНКА КАЧЕСТВА ПРОГРАММНЫХ СРЕДСТВ.	Самостоятельная работа обучающихся №8. Разработка тестового сценария.	2
	Самостоятельная работа обучающихся №9. Оценка необходимого количества тестов.	2
	Самостоятельная работа обучающихся №10. Разработка тестовых пакетов.	2
	Самостоятельная работа обучающихся №11. Оценка программных средств с помощью метрик.	2
	Самостоятельная работа обучающихся №12. Инспекция программного кода на предмет соответствия стандартам кодирования	2

Самостоятельная работа №1. Анализ предметной области

Тема типовая: Разработка системы "Умная библиотека", вам нужно взять из Приложения 1.

Задание:

1. Изучите современные библиотечные системы (как традиционные, так и цифровые)
2. Проведите анализ следующих аспектов:
   - Ключевые пользователи системы (не менее 5 типов)
   - Основные бизнес-процессы библиотеки
   - Проблемы существующих систем
   - Тренды в библиотечном деле
3. Создайте документ "Анализ предметной области" (3-4 страницы), включающий:
   - Описание предметной области
   - Стейкхолдеров и их интересы
   - Основные функциональные потребности
   - Ограничения и риски

Критерии оценки:

- Полнота анализа пользователей
- Глубина понимания бизнес-процессов
- Выявление современных трендов
- Качество оформления документа

Самостоятельная работа №2. Разработка и оформление технического задания

Задание типовое вам нужно по своей предметной области: На основе анализа вашей предметной области разработайте Техническое Задание (ТЗ) на систему.

Требования к ТЗ:

1. Соответствие ГОСТ 34.602-89
2. Объем: 5-7 страниц
3. Обязательные разделы:
   - Назначение и цели создания системы

- Требования к функционалу (не менее 15 функций)
- Требования к пользовательскому интерфейсу
- Требования к надежности и безопасности
- Стадии и этапы разработки
- Порядок контроля и приемки

Конкретные функциональные требования должны включать:

- Электронный каталог книг
- Систему бронирования
- Рекомендательную систему
- Личный кабинет читателя
- Модуль учета посещений
- Систему штрафов и напоминаний

Критерии оценки:

- Полнота и структурированность ТЗ
- Конкретность требований
- Соответствие стандартам оформления

Самостоятельная работа №3. Изучение работы в системе контроля версий

Задание типовое вам нужно по своей предметной области:

Создайте учетную запись на GitHub/GitLab

1. Создайте репозиторий smart-library-system
2. Организуйте структуру репозитория:

text

```
/docs          # документация
/src           # исходный код
/tests         # тесты
/diagrams      # диаграммы
README.md      # описание проекта
.gitignore     # игнорируемые файлы
```

Требуемые действия с Git:

1. Создайте ветку develop от main
2. В ветке develop создайте структуру каталогов
3. Создайте файл README.md с описанием проекта

4. Создайте минимум 5 коммитов с осмысленными сообщениями
5. Создайте ветку `feature/user-auth` и добавьте туда файл `auth_requirements.md`
6. Смержите ветку `feature/user-auth` в `develop`
7. Создайте Pull Request из `develop` в `main`
8. Добавьте тег `v0.1.0`

Критерии оценки:

- Корректность команд Git
- Качество сообщений коммитов
- Организация структуры репозитория
- Использование ветвления

Самостоятельная работа №4. Построение диаграммы Вариантов использования и диаграммы последовательности

Задание типовое вам нужно по своей предметной области:

1. Создайте диаграмму вариантов использования (Use Case Diagram) для следующих акторов:
   - Читатель
   - Библиотекарь
   - Администратор системы
   - Гость (неавторизованный пользователь)
   - Внешняя платежная система
2. Детализируйте 3 ключевых сценария с помощью диаграмм последовательности (Sequence Diagram):
   - Регистрация нового читателя
   - Поиск и бронирование книги
   - Формирование отчета по популярным книгам за месяц

Требования:

- Использовать инструмент PlantUML или аналогичный
- Не менее 10 прецедентов на Use Case Diagram
- Каждая диаграмма последовательности должна содержать минимум 5 объектов
- Сохранить исходный код диаграмм в репозитории

Критерии оценки:

- Полнота охвата функционала
- Корректность UML-нотации
- Логичность взаимодействий

Самостоятельная работа №5. Построение диаграммы Кооперации и диаграммы развертывания

Задание типовое вам нужно по своей предметной области:

Создайте диаграмму кооперации (Collaboration Diagram) для сценария:

1. "Читатель продлевает срок возврата книги через личный кабинет"
2. Создайте диаграмму развертывания (Deployment Diagram) системы "Умная библиотека":
   - Веб-сервер (Nginx/Apache)
   - Сервер приложений (PHP/Python)
   - База данных (MySQL/PostgreSQL)
   - Кэш-сервер (Redis)
   - Файловое хранилище
   - Клиентские устройства (ПК, смартфоны, терминалы)

Требования:

- Диаграмма кооперации: минимум 6 объектов, пронумерованные сообщения
- Диаграмма развертывания: указать протоколы взаимодействия между узлами
- Описать характеристики каждого узла (ОС, ПО)

Критерии оценки:

- Детализация взаимодействий
- Реалистичность архитектуры развертывания
- Комплексность решения

Самостоятельная работа №6. Построение диаграммы Деятельности, диаграммы состояний и диаграммы классов

Задание типовое вам нужно по своей предметной области:

1. **Диаграмма деятельности (Activity Diagram):** Процесс "Обработка поступления новой партии книг в библиотеку"

2. **Диаграмма состояний (State Machine Diagram):** Жизненный цикл объекта "Книга" (статусы: заказана, получена, каталогизирована, доступна, выдана, забронирована, списана)

3. **Диаграмма классов (Class Diagram):** Основные сущности системы (минимум 8 классов с атрибутами и методами)

Требования к диаграмме классов:

- Классы: Book, Reader, Librarian, Loan, Reservation, Category, Author, Penalty
- Указать типы данных атрибутов
- Отобразить отношения: наследование, агрегация, композиция, ассоциация
- Указать видимость методов и атрибутов

Критерии оценки:

- Корректность моделирования процессов
- Полнота состояний объекта
- Связность и непротиворечивость классов

Самостоятельная работа №7. Построение диаграмм потоков данных

Задание типовое вам нужно по своей предметной области: Создайте многоуровневую DFD (Data Flow Diagram) для процесса "Обслуживание читателя":

1. **Контекстная диаграмма (уровень 0):** Система "Умная библиотека" и внешние сущности

2. **Диаграмма верхнего уровня (уровень 1):** Основные процессы системы

3. **Детализация процесса (уровень 2):** Процесс "Выдача книги читателю" (декомпозиция на подпроцессы)

Требования:

- Использовать нотацию Гейна-Сарсона
- Минимум 5 внешних сущностей
- Минимум 2 хранилища данных
- Показать минимум 10 потоков данных

Критерии оценки:

- Правильность декомпозиции
- Полнота потоков данных

- Четкость разделения процессов

Самостоятельная работа №8. Разработка тестового сценария

Задание типовое вам нужно по своей предметной области: Разработайте тестовые сценарии для модуля "Бронирование книги":

1. **Формат сценариев:** Использовать шаблон Gherkin (Given-When-Then)
2. **Количество:** 10 сценариев (7 позитивных, 3 негативных)
3. **Пример позитивного сценария:**

text

Scenario: Успешное бронирование доступной книги

Given Читатель авторизован в системе

And Книга "Война и мир" доступна для бронирования

When Читатель выбирает книгу "Война и мир"

And Нажимает кнопку "Забронировать"

Then Система подтверждает бронирование

And Книга отображается в разделе "Забронированные"

And На email читателя приходит уведомление

Дополнительные сценарии должны включать:

- Бронирование уже забронированной книги
- Бронирование читателем с просроченными книгами
- Бронирование книг разных категорий
- Отмену бронирования

Критерии оценки:

- Полнота покрытия функционала
- Четкость формулировок
- Баланс позитивных и негативных сценариев

Самостоятельная работа №9. Оценка необходимого количества тестов

Задание типовое вам нужно по своей предметной области: Для модуля "Аутентификация пользователей" проведите оценку необходимого количества тестов:

1. **Метод эквивалентных классов:**
   - Поля: логин (email), пароль, запомнить меня
   - Выделить классы эквивалентности

- Определить тестовые случаи

2. **Анализ граничных значений:**

- Длина пароля (минимум 8, максимум 64 символа)
- Возраст читателя (от 14 до 100 лет)

3. **Таблица принятия решений:**

- Условия: валидный email, валидный пароль, аккаунт подтвержден
- Действия: успешный вход, ошибка валидации, требование подтверждения

4. **Рассчитайте минимальное необходимое количество тестов**

Требования:

- Представить расчеты в табличном виде
- Обосновать выбор каждого теста
- Создать матрицу покрытия требований тестами

Критерии оценки:

- Корректность применения методов
- Полнота покрытия условий
- Эффективность набора тестов

Самостоятельная работа №10. Разработка тестовых пакетов

Задание типовое, вам нужно по предлагаемому коду php, Pyton (Приложении 2):

Разработайте автоматизированные тесты:

Часть 1: Модульные тесты (PHPUnit)

php

```
// Тесты для класса Book
class BookTest extends TestCase
{
    public function testCanCreateBookWithValidData() { ... }
    public function testCannotCreateBookWithEmptyTitle() { ... }
    public function testBookStatusChangesCorrectly() { ... }
    // Всего 8 тестов
}
```

Часть 2: Интеграционные тесты

- Тестирование взаимодействия BookRepository с базой данных
- Тестирование поиска книг по различным критериям

Часть 3: Настройка тестового окружения

1. Создать phpunit.xml конфигурацию
2. Настроить тестовую базу данных
3. Создать фикстуры (тестовые данные)

Требования:

- Минимум 15 тестов
- Покрытие кода не менее 70%
- Использование мок-объектов для зависимостей
- Настройка CI/CD pipeline для запуска тестов

Критерии оценки:

- Качество тестового кода
- Полнота покрытия функционала
- Настройка автоматического выполнения

Самостоятельная работа №11. Оценка программных средств с помощью метрик

Задание типовое, по предлагаемому коду php, Pyton (Приложении 2) оценить с помощью метрик программный код.

Пример выполнения.

Проанализируйте предоставленный код модуля "Каталог книг" с помощью метрик:

Исходный код для анализа:

php

```
class BookCatalog {
    private $books = [];

    public function addBook($title, $author, $year, $category,
                            $isbn, $pages, $publisher, $language,
                            $description, $keywords, $price, $count) {
        // Сложный метод с многими параметрами
    }
```

```
    public function findBooks($criteria) {
        // Метод с высокой цикломатической сложностью
        if (isset($criteria['title'])) {
            // ... 20 строк кода с вложенными условиями
        }
        // Еще 50 строк кода
    }

    // Еще 8 методов класса
}
```

Задачи:

1. Рассчитайте вручную для метода findBooks():
   - Цикломатическую сложность
   - Количество строк кода (SLOC)
   - Индекс поддерживаемости
2. Установите и настройте инструменты:
   - PHPMD для анализа проблем кода
   - PHP Metrics для расчета метрик
3. Сгенерируйте отчеты:
   - Отчет о нарушении стандартов кода
   - Отчет с метриками (сложность, связность, наследование)
4. Предложите рефакторинг на основе метрик

Критерии оценки:

- Точность расчетов метрик
- Качество анализа инструментами
- Обоснованность рекомендаций по рефакторингу

Самостоятельная работа №12. Инспекция программного кода на предмет соответствия стандартам кодирования.

Задание типовое, по предлагаемому коду php, Pyton (Приложении 2), необходимо провести инспекцию программного кода, на предмет соответствия стандартам кодирования.

Пример выполнения.

Проведите полную инспекцию кода системы "Умная библиотека":

Часть 1: Статический анализ

1. Установите и настройте:
   - PHP_CodeSniffer с правилами PSR-12
   - PHPStan для статического анализа типов
   - Psalm для поиска ошибок
2. Проанализируйте 5 файлов системы
3. Создайте отчет о нарушениях с классификацией:
   - Критические ошибки (безопасность, функциональность)
   - Основные нарушения (стандарты кодирования)
   - Предупреждения (стиль, возможные улучшения)

Часть 2: Ручная инспекция Проверьте код на соответствие:

1. **Стандартам именования:** PSR-1, PSR-12
2. **Архитектурным принципам:** SOLID, DRY, KISS
3. **Безопасности:** SQL-инъекции, XSS, CSRF
4. **Производительности:** N+1 проблема, оптимизация запросов

Часть 3: Создание правил для проекта Разработайте файл `ruleset.xml` для PHP_CodeSniffer с кастомными правилами:

- Максимальная длина строки: 100 символов
- Запрет на использование `var_dump()`
- Требование type hints для всех методов
- Правила для именования тестовых методов

Требования:

- Проверить минимум 500 строк кода
- Составить план исправления нарушений
- Создать скрипт для автоматической проверки перед коммитом

Критерии оценки:

- Полнота проверки
- Глубина анализа
- Практическая ценность рекомендаций
- Качество настроенных правил

Интеграция всех работ в единый проект

Итоговый результат:

Студент создает полный пакет документации и кода для системы "Умная библиотека", включающий:

1. **Документация:**
   - Анализ предметной области
   - Техническое задание
   - Полный набор UML-диаграмм
   - Тестовая документация

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

Основная литература

1. Рудаков А.В. Технология разработки программных продуктов: учебное издание / Рудаков А.В. - Москва : Академия, 2024. - 208 с. (Специальности среднего профессионального образования). - URL: https://academia-moscow.ru - Режим доступа: Электронная библиотека «Academiamoscow». - Текст : электронный.

Дополнительная литература

1. Казанский, А. А. Объектно-ориентированное программирование. Visual Basic : учебник для среднего профессионального образования / А. А. Казанский. — 2-е изд. — Москва : Издательство Юрайт, 2025. — 295 с. — (Профессиональное образование). — ISBN 978-5-534-21384-3. — Текст : электронный // Образовательная платформа Юрайт [сайт]. — URL: https://urait.ru/bcode/569868.

2. Казанский, А. А. Программирование на visual c# 2013.: учебное пособие для СПО / Казанский А. А.. – Москва : Юрайт, 2020. – 191 с. – ISBN 978-5-534-02721-1. – URL: https://urait.ru/book/programmirovaniena-visual-c-2013-452454. – Текст : электронный.

ПРИЛОЖЕНИЕ 1. Список предметных областей

1. Разработка системы "Электронный университет" (управление студентами, курсами, расписанием).
2. Разработка системы "Умная больница" (учет пациентов, запись к врачам, медицинские карты).
3. Разработка системы "Онлайн-кинотеатр" (каталог фильмов, подписки, рекомендации).
4. Разработка системы "Фитнес-трекер" (учет тренировок, питание, прогресс).
5. Разработка системы "Умный дом" (управление устройствами, сценарии, энергопотребление).
6. Разработка системы "Онлайн-магазин" (каталог товаров, корзина, заказы, доставка).
7. Разработка системы "Бронирование отелей" (поиск, бронирование, управление номерами).
8. Разработка системы "Агрегатор новостей" (сбор новостей из разных источников, персонализация).
9. Разработка системы "Платформа для онлайн-обучения" (курсы, уроки, тесты, сертификаты).
10. Разработка системы "Управление проектами" (задачи, команды, время, отчеты).
11. Разработка системы "Социальная сеть для фотографов" (лента, комментарии, конкурсы).
12. Разработка системы "Форум разработчиков" (темы, ответы, рейтинги, метки).
13. Разработка системы "Система поддержки клиентов" (тикеты, чат, база знаний).
14. Разработка системы "Учет финансов" (доходы, расходы, категории, отчеты).
15. Разработка системы "Планировщик путешествий" (маршруты, бронирование, достопримечательности).
16. Разработка системы "Резюме и вакансии" (поиск работы, отклики, тесты)

17. Разработка системы "Онлайн-резервирование ресторанов" (столики, меню, отзывы).

18. Разработка системы "Каршеринг" (аренда автомобилей, оплата, парковки).

19. Разработка системы "Платформа для мероприятий" (анонсы, билеты, регистрация).

20. Разработка системы "Умная ферма" (мониторинг растений, автоматический полив, отчеты).

21. Разработка системы "Система контроля доступа" (учет сотрудников, пропуска, время).

22. Разработка системы "Чат-бот для заказа еды" (интеграция с мессенджерами, заказ, оплата).

23. Разработка системы "Платформа для краудфандинга" (проекты, сбор средств, отчеты).

24. Разработка системы "Система мониторинга транспорта" (GPS-трекеры, маршруты, расход топлива).

25. Разработка системы "Онлайн-конструктор резюме" (шаблоны, заполнение, экспорт).

26. Разработка системы "Платформа для стриминга" (видео, подписки, донаты).

27. Разработка системы "Управление складом" (учет товаров, перемещения, инвентаризация).

28. Разработка системы "Погодный мониторинг" (данные с датчиков, прогнозы, оповещения).

29. Разработка системы "Спортивный трекер" (результаты соревнований, статистика, рейтинги) .

30. Своя тема (согласование с преподавателем).

ПРИЛОЖЕНИЕ 2. Код для анализа.

PHP:

Структура пректа.

```
smart-library/
├── src/
│   ├── Models/
│   │   ├── Book.php
│   │   ├── Reader.php
│   │   └── Loan.php
│   ├── Repositories/
│   │   ├── BookRepository.php
│   │   └── CatalogRepository.php
│   ├── Services/
│   │   ├── CatalogService.php
│   │   └── SearchService.php
│   └── Utils/
│       └── Validator.php
│
├── tests/
│   ├── Unit/
│   │   ├── Models/
│   │   │   ├── BookTest.php
│   │   │   └── ReaderTest.php
│   │   ├── Services/
│   │   │   ├── CatalogServiceTest.php
│   │   │   └── SearchServiceTest.php
│   │   └── Utils/
│   │       └── ValidatorTest.php
│   ├── Integration/
│   │   ├── BookRepositoryTest.php
│   │   └── CatalogRepositoryTest.php
│   ├── Fixtures/
│   │   └── TestData.php
│   └── bootstrap.php
│
├── config/
│   ├── database.php
│   └── test_database.php
│
├── reports/
│   ├── coverage/
│   ├── metrics/
│   └── phpmd/
│
├── phpunit.xml
├── composer.json
├── .gitlab-ci.yml
└── README.md
```

1. Исходный код для тестирования

src/Models/Book.php

php

```php
<?php

namespace SmartLibrary\Models;

/**
 * Модель книги
 */
class Book
{
    private const STATUS_AVAILABLE = 'available';
    private const STATUS_BORROWED = 'borrowed';
    private const STATUS_RESERVED = 'reserved';
    private const STATUS_ARCHIVED = 'archived';

    private int $id;
    private string $title;
    private string $author;
    private int $publicationYear;
    private string $isbn;
    private string $category;
    private int $pages;
    private string $publisher;
    private string $language;
    private string $description;
    private array $keywords;
    private float $price;
    private int $totalCopies;
    private int $availableCopies;
```

```
    private string $status;
    private \DateTime $createdAt;
    private ?\DateTime $updatedAt;

    public function __construct(
        string $title,
        string $author,
        int $publicationYear,
        string $isbn,
        string $category,
        int $pages,
        string $publisher,
        string $language,
        string $description,
        array $keywords,
        float $price,
        int $totalCopies
    ) {
        $this->validateConstructorParams(
            $title, $author, $publicationYear, $isbn, $category,
            $pages, $publisher, $language, $description, $keywords,
            $price, $totalCopies
        );

        $this->title = $title;
        $this->author = $author;
        $this->publicationYear = $publicationYear;
        $this->isbn = $isbn;
        $this->category = $category;
        $this->pages = $pages;
        $this->publisher = $publisher;
        $this->language = $language;
```

```
        $this->description = $description;
        $this->keywords = $keywords;
        $this->price = $price;
        $this->totalCopies = $totalCopies;
        $this->availableCopies = $totalCopies;
        $this->status = self::STATUS_AVAILABLE;
        $this->createdAt = new \DateTime();
        $this->updatedAt = null;
    }

    private function validateConstructorParams(
        string $title,
        string $author,
        int $publicationYear,
        string $isbn,
        string $category,
        int $pages,
        string $publisher,
        string $language,
        string $description,
        array $keywords,
        float $price,
        int $totalCopies
    ): void {
        if (empty($title)) {
            throw new \InvalidArgumentException('Title cannot be empty');
        }

        if (empty($author)) {
            throw new \InvalidArgumentException('Author cannot be empty');
        }
```

```
        if ($publicationYear < 1000 || $publicationYear > (int)date('Y') + 1) {
            throw new \InvalidArgumentException(
                'Publication year must be between 1000 and ' . (date('Y') + 1)
            );
        }

        if (!$this->validateIsbn($isbn)) {
            throw new \InvalidArgumentException('Invalid ISBN format');
        }

        if ($pages <= 0) {
            throw new \InvalidArgumentException('Pages must be positive');
        }

        if ($price < 0) {
            throw new \InvalidArgumentException('Price cannot be negative');
        }

        if ($totalCopies <= 0) {
            throw new \InvalidArgumentException('Total copies must be positive');
        }
    }

    private function validateIsbn(string $isbn): bool
    {
        // Упрощенная валидация ISBN (10 или 13 цифр)
        $isbn = str_replace(['-', ' '], '', $isbn);
        return preg_match('/^(?:\d{9}[\dX]|\d{13})$/', $isbn);
    }

    public function borrow(): void
    {
```

```
    if ($this->availableCopies <= 0) {
        throw new \RuntimeException('No copies available for borrowing');
    }

    if ($this->status === self::STATUS_ARCHIVED) {
        throw new \RuntimeException('Cannot borrow archived book');
    }

    $this->availableCopies--;
    $this->updateStatus();
    $this->updatedAt = new \DateTime();
}

public function return(): void
{
    if ($this->availableCopies >= $this->totalCopies) {
        throw new \RuntimeException('All copies are already available');
    }

    $this->availableCopies++;
    $this->updateStatus();
    $this->updatedAt = new \DateTime();
}

public function reserve(): void
{
    if ($this->availableCopies <= 0) {
        throw new \RuntimeException('Cannot reserve book with no available copies');
    }

    $this->status = self::STATUS_RESERVED;
    $this->updatedAt = new \DateTime();
```

```
}

public function cancelReservation(): void
{
    if ($this->status !== self::STATUS_RESERVED) {
        throw new \RuntimeException('Book is not reserved');
    }

    $this->updateStatus();
    $this->updatedAt = new \DateTime();
}

private function updateStatus(): void
{
    if ($this->availableCopies <= 0) {
        $this->status = self::STATUS_BORROWED;
    } elseif ($this->availableCopies < $this->totalCopies) {
        $this->status = self::STATUS_BORROWED;
    } else {
        $this->status = self::STATUS_AVAILABLE;
    }
}

public function archive(): void
{
    if ($this->availableCopies !== $this->totalCopies) {
        throw new \RuntimeException('Cannot archive book with borrowed copies');
    }

    $this->status = self::STATUS_ARCHIVED;
    $this->updatedAt = new \DateTime();
}
```

```php
// Геттеры
public function getId(): int { return $this->id; }
public function getTitle(): string { return $this->title; }
public function getAuthor(): string { return $this->author; }
public function getPublicationYear(): int { return $this->publicationYear; }
public function getIsbn(): string { return $this->isbn; }
public function getCategory(): string { return $this->category; }
public function getPages(): int { return $this->pages; }
public function getPublisher(): string { return $this->publisher; }
public function getLanguage(): string { return $this->language; }
public function getDescription(): string { return $this->description; }
public function getKeywords(): array { return $this->keywords; }
public function getPrice(): float { return $this->price; }
public function getTotalCopies(): int { return $this->totalCopies; }
public function getAvailableCopies(): int { return $this->availableCopies; }
public function getStatus(): string { return $this->status; }
public function getCreatedAt(): \DateTime { return $this->createdAt; }
public function getUpdatedAt(): ?\DateTime { return $this->updatedAt; }

// Сеттеры с валидацией
public function setTitle(string $title): void
{
    if (empty($title)) {
        throw new \InvalidArgumentException('Title cannot be empty');
    }
    $this->title = $title;
    $this->updatedAt = new \DateTime();
}

public function setAvailableCopies(int $copies): void
{
```

```
    if ($copies < 0 || $copies > $this->totalCopies) {
        throw new \InvalidArgumentException(
            "Available copies must be between 0 and {$this->totalCopies}"
        );
    }
    $this->availableCopies = $copies;
    $this->updateStatus();
    $this->updatedAt = new \DateTime();
}

public function toArray(): array
{
    return [
        'id' => $this->id,
        'title' => $this->title,
        'author' => $this->author,
        'publication_year' => $this->publicationYear,
        'isbn' => $this->isbn,
        'category' => $this->category,
        'pages' => $this->pages,
        'publisher' => $this->publisher,
        'language' => $this->language,
        'description' => $this->description,
        'keywords' => $this->keywords,
        'price' => $this->price,
        'total_copies' => $this->totalCopies,
        'available_copies' => $this->availableCopies,
        'status' => $this->status,
        'created_at' => $this->createdAt->format('Y-m-d H:i:s'),
        'updated_at' => $this->updatedAt ? $this->updatedAt->format('Y-m-d H:i:s') : null,
    ];
}
```

}

src/Repositories/BookRepository.php

php

```php
<?php

namespace SmartLibrary\Repositories;

use SmartLibrary\Models\Book;
use PDO;
use PDOException;

/**
 * Репозиторий для работы с книгами в базе данных
 */
class BookRepository
{
    private PDO $connection;

    public function __construct(PDO $connection)
    {
        $this->connection = $connection;
    }

    public function save(Book $book): int
    {
        $sql = "
            INSERT INTO books (
                title, author, publication_year, isbn, category,
                pages, publisher, language, description, keywords,
                price, total_copies, available_copies, status
            ) VALUES (
                :title, :author, :publication_year, :isbn, :category,
```

```
            :pages, :publisher, :language, :description, :keywords,
            :price, :total_copies, :available_copies, :status
        )
    ";

    $stmt = $this->connection->prepare($sql);

    $data = $book->toArray();
    unset($data['id'], $data['created_at'], $data['updated_at']);
    $data['keywords'] = json_encode($data['keywords']);

    $stmt->execute($data);

    return (int)$this->connection->lastInsertId();
}

public function findById(int $id): ?Book
{
    $sql = "SELECT * FROM books WHERE id = :id";
    $stmt = $this->connection->prepare($sql);
    $stmt->execute([':id' => $id]);

    $data = $stmt->fetch(PDO::FETCH_ASSOC);

    if (!$data) {
        return null;
    }

    return $this->hydrate($data);
}

public function findByIsbn(string $isbn): ?Book
```

```
{
    $sql = "SELECT * FROM books WHERE isbn = :isbn";
    $stmt = $this->connection->prepare($sql);
    $stmt->execute([':isbn' => $isbn]);

    $data = $stmt->fetch(PDO::FETCH_ASSOC);

    if (!$data) {
        return null;
    }

    return $this->hydrate($data);
}

public function findAll(int $limit = 100, int $offset = 0): array
{
    $sql = "SELECT * FROM books ORDER BY title LIMIT :limit OFFSET :offset";
    $stmt = $this->connection->prepare($sql);
    $stmt->bindValue(':limit', $limit, PDO::PARAM_INT);
    $stmt->bindValue(':offset', $offset, PDO::PARAM_INT);
    $stmt->execute();

    $books = [];
    while ($data = $stmt->fetch(PDO::FETCH_ASSOC)) {
        $books[] = $this->hydrate($data);
    }

    return $books;
}

public function update(Book $book): bool
{
```

```
        $sql = "
            UPDATE books SET
                title = :title,
                author = :author,
                publication_year = :publication_year,
                isbn = :isbn,
                category = :category,
                pages = :pages,
                publisher = :publisher,
                language = :language,
                description = :description,
                keywords = :keywords,
                price = :price,
                total_copies = :total_copies,
                available_copies = :available_copies,
                status = :status,
                updated_at = NOW()
            WHERE id = :id
        ";

        $stmt = $this->connection->prepare($sql);

        $data = $book->toArray();
        $data['keywords'] = json_encode($data['keywords']);

        return $stmt->execute($data);
    }

    public function delete(int $id): bool
    {
        $sql = "DELETE FROM books WHERE id = :id";
        $stmt = $this->connection->prepare($sql);
```

```
    return $stmt->execute([':id' => $id]);
}

public function search(array $criteria): array
{
    $sql = "SELECT * FROM books WHERE 1=1";
    $params = [];

    if (!empty($criteria['title'])) {
        $sql .= " AND title LIKE :title";
        $params[':title'] = '%' . $criteria['title'] . '%';
    }

    if (!empty($criteria['author'])) {
        $sql .= " AND author LIKE :author";
        $params[':author'] = '%' . $criteria['author'] . '%';
    }

    if (!empty($criteria['category'])) {
        $sql .= " AND category = :category";
        $params[':category'] = $criteria['category'];
    }

    if (!empty($criteria['min_year'])) {
        $sql .= " AND publication_year >= :min_year";
        $params[':min_year'] = $criteria['min_year'];
    }

    if (!empty($criteria['max_year'])) {
        $sql .= " AND publication_year <= :max_year";
        $params[':max_year'] = $criteria['max_year'];
    }
```

```
if (!empty($criteria['min_pages'])) {
    $sql .= " AND pages >= :min_pages";
    $params[':min_pages'] = $criteria['min_pages'];
}

if (!empty($criteria['max_pages'])) {
    $sql .= " AND pages <= :max_pages";
    $params[':max_pages'] = $criteria['max_pages'];
}

if (isset($criteria['available_only']) && $criteria['available_only']) {
    $sql .= " AND available_copies > 0";
}

if (!empty($criteria['keywords'])) {
    $keywords = $criteria['keywords'];
    if (is_string($keywords)) {
        $keywords = explode(',', $keywords);
    }

    $keywordConditions = [];
    foreach ($keywords as $i => $keyword) {
        $param = ':keyword_' . $i;
        $keywordConditions[] = "keywords LIKE {$param}";
        $params[$param] = '%' . trim($keyword) . '%';
    }

    if (!empty($keywordConditions)) {
        $sql .= " AND (" . implode(' OR ', $keywordConditions) . ")";
    }
}
```

```
        $sql .= " ORDER BY title LIMIT 100";

        $stmt = $this->connection->prepare($sql);
        $stmt->execute($params);

        $books = [];
        while ($data = $stmt->fetch(PDO::FETCH_ASSOC)) {
            $books[] = $this->hydrate($data);
        }

        return $books;
    }

    public function countByCategory(string $category): int
    {
        $sql = "SELECT COUNT(*) as count FROM books WHERE category = :category";
        $stmt = $this->connection->prepare($sql);
        $stmt->execute([':category' => $category]);

        $result = $stmt->fetch(PDO::FETCH_ASSOC);
        return (int)($result['count'] ?? 0);
    }

    public function getTotalBooksCount(): int
    {
        $sql = "SELECT COUNT(*) as count FROM books";
        $stmt = $this->connection->query($sql);
        $result = $stmt->fetch(PDO::FETCH_ASSOC);
        return (int)($result['count'] ?? 0);
    }
```

```php
private function hydrate(array $data): Book
{
    $book = new Book(
        $data['title'],
        $data['author'],
        (int)$data['publication_year'],
        $data['isbn'],
        $data['category'],
        (int)$data['pages'],
        $data['publisher'],
        $data['language'],
        $data['description'],
        json_decode($data['keywords'], true) ?? [],
        (float)$data['price'],
        (int)$data['total_copies']
    );

    // Устанавливаем свойства, которые не в конструкторе
    $reflection = new \ReflectionClass($book);

    $idProperty = $reflection->getProperty('id');
    $idProperty->setAccessible(true);
    $idProperty->setValue($book, (int)$data['id']);

    $availableCopiesProperty = $reflection->getProperty('availableCopies');
    $availableCopiesProperty->setAccessible(true);
    $availableCopiesProperty->setValue($book, (int)$data['available_copies']);

    $statusProperty = $reflection->getProperty('status');
    $statusProperty->setAccessible(true);
    $statusProperty->setValue($book, $data['status']);
```

```
        $createdAtProperty = $reflection->getProperty('createdAt');
        $createdAtProperty->setAccessible(true);
        $createdAtProperty->setValue($book, new \DateTime($data['created_at']));

        if (!empty($data['updated_at'])) {
            $updatedAtProperty = $reflection->getProperty('updatedAt');
            $updatedAtProperty->setAccessible(true);
            $updatedAtProperty->setValue($book, new \DateTime($data['updated_at']));
        }

        return $book;
    }
}
```

src/Services/CatalogService.php (класс с проблемами для анализа метрик)

```
php
<?php

namespace SmartLibrary\Services;

use SmartLibrary\Repositories\BookRepository;
use SmartLibrary\Models\Book;

/**
 * Сервис каталога книг (специально с проблемами для анализа метрик)
 */
class CatalogService
{
    private BookRepository $bookRepository;
    private array $statisticsCache = [];
    private int $cacheTtl = 3600;

    public function __construct(BookRepository $bookRepository)
```

```
{
    $this->bookRepository = $bookRepository;
}

/**
 * Метод с высокой цикломатической сложностью и многими параметрами
 */
public function addBook(
    string $title,
    string $author,
    int $publicationYear,
    string $isbn,
    string $category,
    int $pages = 0,
    string $publisher = '',
    string $language = 'Russian',
    string $description = '',
    array $keywords = [],
    float $price = 0.0,
    int $totalCopies = 1,
    bool $validateIsbn = true,
    bool $checkDuplicates = true,
    bool $updateStatistics = true
): Book {
    // Валидация названия
    if (empty(trim($title))) {
        throw new \InvalidArgumentException('Book title cannot be empty');
    }

    if (strlen($title) < 2) {
        throw new \InvalidArgumentException('Book title is too short');
    }
```

```php
if (strlen($title) > 255) {
    throw new \InvalidArgumentException('Book title is too long');
}

// Валидация автора
if (empty(trim($author))) {
    throw new \InvalidArgumentException('Author cannot be empty');
}

// Валидация года издания
$currentYear = (int)date('Y');
if ($publicationYear < 1000) {
    throw new \InvalidArgumentException('Publication year is too early');
}

if ($publicationYear > $currentYear + 1) {
    throw new \InvalidArgumentException('Publication year is in the future');
}

// Валидация ISBN если требуется
if ($validateIsbn) {
    if (!$this->validateIsbnFormat($isbn)) {
        throw new \InvalidArgumentException('Invalid ISBN format');
    }
}

// Проверка на дубликаты если требуется
if ($checkDuplicates) {
    $existingBook = $this->bookRepository->findByIsbn($isbn);
    if ($existingBook !== null) {
        throw new \RuntimeException('Book with this ISBN already exists');
```

```
    }
}

// Валидация категории
$validCategories = $this->getValidCategories();
if (!in_array($category, $validCategories, true)) {
    throw new \InvalidArgumentException('Invalid book category');
}

// Валидация количества страниц
if ($pages < 0) {
    throw new \InvalidArgumentException('Pages cannot be negative');
}

if ($pages > 10000) {
    throw new \InvalidArgumentException('Too many pages');
}

// Валидация цены
if ($price < 0) {
    throw new \InvalidArgumentException('Price cannot be negative');
}

if ($price > 1000000) {
    throw new \InvalidArgumentException('Price is too high');
}

// Валидация количества копий
if ($totalCopies <= 0) {
    throw new \InvalidArgumentException('Total copies must be positive');
}
```

```php
    if ($totalCopies > 1000) {
        throw new \InvalidArgumentException('Too many copies');
    }

    // Создание книги
    $book = new Book(
        $title,
        $author,
        $publicationYear,
        $isbn,
        $category,
        $pages,
        $publisher,
        $language,
        $description,
        $keywords,
        $price,
        $totalCopies
    );

    // Сохранение в репозитории
    $bookId = $this->bookRepository->save($book);

    // Обновление статистики если требуется
    if ($updateStatistics) {
        $this->updateCategoryStatistics($category);
        $this->clearStatisticsCache();
    }

    return $book;
}
```

```php
/**
 * Сложный метод поиска с множеством условий
 */
public function findBooks(array $criteria): array
{
    $results = [];

    // Если есть критерии поиска
    if (!empty($criteria)) {
        // Поиск по названию (точное совпадение)
        if (isset($criteria['title_exact']) && !empty($criteria['title_exact'])) {
            $books = $this->bookRepository->search(['title' => $criteria['title_exact']]);
            $results = array_merge($results, $books);
        }

        // Поиск по названию (частичное совпадение)
        if (isset($criteria['title_like']) && !empty($criteria['title_like'])) {
            $books = $this->bookRepository->search(['title' => $criteria['title_like']]);
            foreach ($books as $book) {
                if (!$this->isBookInResults($book, $results)) {
                    $results[] = $book;
                }
            }
        }

        // Поиск по автору
        if (isset($criteria['author']) && !empty($criteria['author'])) {
            $books = $this->bookRepository->search(['author' => $criteria['author']]);
            foreach ($books as $book) {
                if (!$this->isBookInResults($book, $results)) {
                    $results[] = $book;
                }
```

```
    }
}

// Поиск по категории
if (isset($criteria['category']) && !empty($criteria['category'])) {
    $books = $this->bookRepository->search(['category' => $criteria['category']]);
    foreach ($books as $book) {
        if (!$this->isBookInResults($book, $results)) {
            $results[] = $book;
        }
    }
}

// Поиск по году издания
if (isset($criteria['year_from']) || isset($criteria['year_to'])) {
    $yearCriteria = [];
    if (isset($criteria['year_from'])) {
        $yearCriteria['min_year'] = $criteria['year_from'];
    }
    if (isset($criteria['year_to'])) {
        $yearCriteria['max_year'] = $criteria['year_to'];
    }

    $books = $this->bookRepository->search($yearCriteria);
    foreach ($books as $book) {
        if (!$this->isBookInResults($book, $results)) {
            $results[] = $book;
        }
    }
}

// Поиск по количеству страниц
```

```php
if (isset($criteria['pages_from']) || isset($criteria['pages_to'])) {
    $pagesCriteria = [];
    if (isset($criteria['pages_from'])) {
        $pagesCriteria['min_pages'] = $criteria['pages_from'];
    }
    if (isset($criteria['pages_to'])) {
        $pagesCriteria['max_pages'] = $criteria['pages_to'];
    }

    $books = $this->bookRepository->search($pagesCriteria);
    foreach ($books as $book) {
        if (!$this->isBookInResults($book, $results)) {
            $results[] = $book;
        }
    }
}

// Поиск только доступных книг
if (isset($criteria['available_only']) && $criteria['available_only']) {
    $availableCriteria = ['available_only' => true];
    $books = $this->bookRepository->search($availableCriteria);

    // Фильтруем результаты, оставляя только доступные
    $filteredResults = [];
    foreach ($results as $book) {
        if ($book->getAvailableCopies() > 0) {
            $filteredResults[] = $book;
        }
    }
    $results = $filteredResults;

    // Добавляем книги из поиска по доступности
```

```
    foreach ($books as $book) {
        if (!$this->isBookInResults($book, $results)) {
            $results[] = $book;
        }
    }
}

// Поиск по ключевым словам
if (isset($criteria['keywords']) && !empty($criteria['keywords'])) {
    $keywordCriteria = ['keywords' => $criteria['keywords']];
    $books = $this->bookRepository->search($keywordCriteria);
    foreach ($books as $book) {
        if (!$this->isBookInResults($book, $results)) {
            $results[] = $book;
        }
    }
}

// Сортировка результатов
if (!empty($results)) {
    $sortField = $criteria['sort_by'] ?? 'title';
    $sortDirection = $criteria['sort_dir'] ?? 'asc';

    usort($results, function ($a, $b) use ($sortField, $sortDirection) {
        $valueA = $this->getSortValue($a, $sortField);
        $valueB = $this->getSortValue($b, $sortField);

        if ($sortDirection === 'asc') {
            return $valueA <=> $valueB;
        } else {
            return $valueB <=> $valueA;
        }
```

```
            });
        }

        // Ограничение количества результатов
        if (isset($criteria['limit']) && $criteria['limit'] > 0) {
            $results = array_slice($results, 0, $criteria['limit']);
        }
    } else {
        // Если критериев нет, возвращаем все книги
        $results = $this->bookRepository->findAll();
    }

    return $results;
}

private function isBookInResults(Book $book, array $results): bool
{
    foreach ($results as $result) {
        if ($result->getId() === $book->getId()) {
            return true;
        }
    }
    return false;
}

private function getSortValue(Book $book, string $field)
{
    switch ($field) {
        case 'title':
            return $book->getTitle();
        case 'author':
            return $book->getAuthor();
```

```
            case 'year':
                return $book->getPublicationYear();
            case 'pages':
                return $book->getPages();
            case 'price':
                return $book->getPrice();
            default:
                return $book->getTitle();
        }
    }

    public function getBookStatistics(): array
    {
        $cacheKey = 'book_statistics';

        if (isset($this->statisticsCache[$cacheKey]) &&
            $this->statisticsCache[$cacheKey]['timestamp'] > time() - $this->cacheTtl) {
            return $this->statisticsCache[$cacheKey]['data'];
        }

        $totalBooks = $this->bookRepository->getTotalBooksCount();
        $categories = $this->getValidCategories();

        $stats = [
            'total_books' => $totalBooks,
            'by_category' => [],
            'by_year' => [],
            'by_availability' => [
                'available' => 0,
                'borrowed' => 0,
                'reserved' => 0,
                'archived' => 0
```

```
        ]
    ];

    foreach ($categories as $category) {
        $count = $this->bookRepository->countByCategory($category);
        $stats['by_category'][$category] = $count;
    }

    // Получаем все книги для анализа
    $allBooks = $this->bookRepository->findAll(1000);

    foreach ($allBooks as $book) {
        // Статистика по годам
        $year = $book->getPublicationYear();
        if (!isset($stats['by_year'][$year])) {
            $stats['by_year'][$year] = 0;
        }
        $stats['by_year'][$year]++;

        // Статистика по доступности
        $status = $book->getStatus();
        if ($status === 'available') {
            $stats['by_availability']['available']++;
        } elseif ($status === 'borrowed') {
            $stats['by_availability']['borrowed']++;
        } elseif ($status === 'reserved') {
            $stats['by_availability']['reserved']++;
        } elseif ($status === 'archived') {
            $stats['by_availability']['archived']++;
        }
    }
```

```
        // Сортируем по годам
        ksort($stats['by_year']);

        $this->statisticsCache[$cacheKey] = [
            'data' => $stats,
            'timestamp' => time()
        ];

        return $stats;
    }

    private function validateIsbnFormat(string $isbn): bool
    {
        // Удаляем дефисы и пробелы
        $isbn = str_replace(['-', ' '], '', $isbn);

        // Проверяем длину
        $length = strlen($isbn);
        if ($length !== 10 && $length !== 13) {
            return false;
        }

        // Проверяем, что все символы цифры, кроме последнего который может быть X для ISBN-10
        if ($length === 10) {
            for ($i = 0; $i < 9; $i++) {
                if (!is_numeric($isbn[$i])) {
                    return false;
                }
            }

            $lastChar = $isbn[9];
```

```
    if (!is_numeric($lastChar) && $lastChar !== 'X' && $lastChar !== 'x') {
        return false;
    }

    // Проверка контрольной суммы для ISBN-10
    $sum = 0;
    for ($i = 0; $i < 9; $i++) {
        $sum += (int)$isbn[$i] * (10 - $i);
    }

    $lastDigit = ($lastChar === 'X' || $lastChar === 'x') ? 10 : (int)$lastChar;
    $sum += $lastDigit;

    return ($sum % 11 === 0);
}

// Проверка для ISBN-13
if ($length === 13) {
    for ($i = 0; $i < 13; $i++) {
        if (!is_numeric($isbn[$i])) {
            return false;
        }
    }

    // Проверка контрольной суммы для ISBN-13
    $sum = 0;
    for ($i = 0; $i < 12; $i++) {
        $weight = ($i % 2 === 0) ? 1 : 3;
        $sum += (int)$isbn[$i] * $weight;
    }

    $checksum = (10 - ($sum % 10)) % 10;
```

```
        return ($checksum === (int)$isbn[12]);
    }

    return false;
}

private function getValidCategories(): array
{
    return [
        'Fiction',
        'Non-Fiction',
        'Science',
        'Technology',
        'History',
        'Biography',
        'Children',
        'Fantasy',
        'Mystery',
        'Romance',
        'Science Fiction',
        'Horror',
        'Poetry',
        'Drama',
        'Comics',
        'Art',
        'Cookbooks',
        'Travel',
        'Religion',
        'Education'
    ];
}
```

```php
private function updateCategoryStatistics(string $category): void
{
    // В реальном приложении здесь была бы логика обновления статистики
    // Для примера просто очищаем кэш
    $this->clearStatisticsCache();
}

private function clearStatisticsCache(): void
{
    $this->statisticsCache = [];
}

// Еще несколько методов для увеличения сложности класса
public function importBooks(array $booksData): array
{
    $results = [
        'success' => 0,
        'failed' => 0,
        'errors' => []
    ];

    foreach ($booksData as $index => $bookData) {
        try {
            $book = $this->addBook(
                $bookData['title'] ?? '',
                $bookData['author'] ?? '',
                $bookData['publication_year'] ?? 0,
                $bookData['isbn'] ?? '',
                $bookData['category'] ?? 'Fiction',
                $bookData['pages'] ?? 0,
                $bookData['publisher'] ?? '',
```

```
                $bookData['language'] ?? 'Russian',
                $bookData['description'] ?? '',
                $bookData['keywords'] ?? [],
                $bookData['price'] ?? 0.0,
                $bookData['total_copies'] ?? 1,
                $bookData['validate_isbn'] ?? true,
                $bookData['check_duplicates'] ?? true,
                false // Не обновлять статистику для каждого импорта
            );

            $results['success']++;
        } catch (\Exception $e) {
            $results['failed']++;
            $results['errors'][] = [
                'index' => $index,
                'error' => $e->getMessage(),
                'data' => $bookData
            ];
        }
    }

    // Обновляем статистику один раз после импорта
    $this->clearStatisticsCache();

    return $results;
}

public function exportBooks(array $criteria = []): array
{
    $books = $this->findBooks($criteria);
    $exportData = [];
```

```
    foreach ($books as $book) {
        $exportData[] = [
            'id' => $book->getId(),
            'title' => $book->getTitle(),
            'author' => $book->getAuthor(),
            'isbn' => $book->getIsbn(),
            'category' => $book->getCategory(),
            'year' => $book->getPublicationYear(),
            'pages' => $book->getPages(),
            'price' => $book->getPrice(),
            'copies' => $book->getTotalCopies(),
            'available' => $book->getAvailableCopies(),
            'status' => $book->getStatus()
        ];
    }

    return $exportData;
}

public function generateReport(string $type, array $options = []): array
{
    switch ($type) {
        case 'category_summary':
            return $this->generateCategorySummaryReport($options);
        case 'yearly_summary':
            return $this->generateYearlySummaryReport($options);
        case 'availability_summary':
            return $this->generateAvailabilitySummaryReport($options);
        case 'detailed':
            return $this->generateDetailedReport($options);
        default:
            throw new \InvalidArgumentException("Unknown report type: {$type}");
```

```
    }
}

private function generateCategorySummaryReport(array $options): array
{
    $stats = $this->getBookStatistics();
    $report = [
        'type' => 'category_summary',
        'generated_at' => date('Y-m-d H:i:s'),
        'data' => $stats['by_category']
    ];

    if (isset($options['include_percentage']) && $options['include_percentage']) {
        $total = array_sum($stats['by_category']);
        foreach ($report['data'] as $category => $count) {
            $report['data'][$category] = [
                'count' => $count,
                'percentage' => $total > 0 ? round(($count / $total) * 100, 2) : 0
            ];
        }
    }

    return $report;
}

private function generateYearlySummaryReport(array $options): array
{
    $stats = $this->getBookStatistics();
    $report = [
        'type' => 'yearly_summary',
        'generated_at' => date('Y-m-d H:i:s'),
        'data' => $stats['by_year']
```

```
        ];

        return $report;
    }

    private function generateAvailabilitySummaryReport(array $options): array
    {
        $stats = $this->getBookStatistics();
        $report = [
            'type' => 'availability_summary',
            'generated_at' => date('Y-m-d H:i:s'),
            'data' => $stats['by_availability']
        ];

        return $report;
    }

    private function generateDetailedReport(array $options): array
    {
        $limit = $options['limit'] ?? 100;
        $books = $this->bookRepository->findAll($limit);

        $report = [
            'type' => 'detailed',
            'generated_at' => date('Y-m-d H:i:s'),
            'total_books' => count($books),
            'books' => []
        ];

        foreach ($books as $book) {
            $report['books'][] = $book->toArray();
        }
```

```
        return $report;
    }
}
```

2. Тестовый пакет

tests/Unit/Models/BookTest.php

php

```
<?php

namespace Tests\Unit\Models;

use PHPUnit\Framework\TestCase;
use SmartLibrary\Models\Book;

class BookTest extends TestCase
{
    public function testCanCreateBookWithValidData(): void
    {
        $book = new Book(
            'Test Book',
            'Test Author',
            2023,
            '978-3-16-148410-0',
            'Fiction',
            300,
            'Test Publisher',
            'English',
            'Test Description',
            ['test', 'book'],
            29.99,
            5
        );
```

```
    $this->assertInstanceOf(Book::class, $book);
    $this->assertEquals('Test Book', $book->getTitle());
    $this->assertEquals('Test Author', $book->getAuthor());
    $this->assertEquals(5, $book->getAvailableCopies());
}

public function testCannotCreateBookWithEmptyTitle(): void
{
    $this->expectException(\InvalidArgumentException::class);
    $this->expectExceptionMessage('Title cannot be empty');

    new Book(
        '',
        'Test Author',
        2023,
        '978-3-16-148410-0',
        'Fiction',
        300,
        'Test Publisher',
        'English',
        'Test Description',
        [],
        29.99,
        5
    );
}

public function testCannotCreateBookWithInvalidYear(): void
{
    $this->expectException(\InvalidArgumentException::class);
```

```
        new Book(
            'Test Book',
            'Test Author',
            500, // Слишком ранний год
            '978-3-16-148410-0',
            'Fiction',
            300,
            'Test Publisher',
            'English',
            'Test Description',
            [],
            29.99,
            5
        );
    }

    public function testBookStatusChangesCorrectly(): void
    {
        $book = new Book(
            'Test Book',
            'Test Author',
            2023,
            '978-3-16-148410-0',
            'Fiction',
            300,
            'Test Publisher',
            'English',
            'Test Description',
            [],
            29.99,
            3
        );
```

```
    // Изначально статус "доступно"
    $this->assertEquals('available', $book->getStatus());

    // После заимствования одной копии
    $book->borrow();
    $this->assertEquals(2, $book->getAvailableCopies());
    $this->assertEquals('borrowed', $book->getStatus());

    // После возврата
    $book->return();
    $this->assertEquals(3, $book->getAvailableCopies());
    $this->assertEquals('available', $book->getStatus());
}

public function testCannotBorrowWhenNoCopiesAvailable(): void
{
    $book = new Book(
        'Test Book',
        'Test Author',
        2023,
        '978-3-16-148410-0',
        'Fiction',
        300,
        'Test Publisher',
        'English',
        'Test Description',
        [],
        29.99,
        1
    );
```

```
    $book->borrow(); // Заимствуем единственную копию

    $this->expectException(\RuntimeException::class);
    $this->expectExceptionMessage('No copies available for borrowing');

    $book->borrow(); // Пытаемся заимствовать еще раз
}

public function testReserveAndCancelReservation(): void
{
    $book = new Book(
        'Test Book',
        'Test Author',
        2023,
        '978-3-16-148410-0',
        'Fiction',
        300,
        'Test Publisher',
        'English',
        'Test Description',
        [],
        29.99,
        2
    );

    $book->reserve();
    $this->assertEquals('reserved', $book->getStatus());

    $book->cancelReservation();
    $this->assertEquals('available', $book->getStatus());
}
```

```php
public function testCannotReserveWhenNoCopiesAvailable(): void
{
    $book = new Book(
        'Test Book',
        'Test Author',
        2023,
        '978-3-16-148410-0',
        'Fiction',
        300,
        'Test Publisher',
        'English',
        'Test Description',
        [],
        29.99,
        1
    );

    $book->borrow(); // Заимствуем единственную копию

    $this->expectException(\RuntimeException::class);
    $this->expectExceptionMessage('Cannot reserve book with no available copies');

    $book->reserve();
}

public function testArchiveBook(): void
{
    $book = new Book(
        'Test Book',
        'Test Author',
        2023,
        '978-3-16-148410-0',
```

```
        'Fiction',
        300,
        'Test Publisher',
        'English',
        'Test Description',
        [],
        29.99,
        3
    );

    $book->archive();
    $this->assertEquals('archived', $book->getStatus());
}

public function testCannotArchiveBookWithBorrowedCopies(): void
{
    $book = new Book(
        'Test Book',
        'Test Author',
        2023,
        '978-3-16-148410-0',
        'Fiction',
        300,
        'Test Publisher',
        'English',
        'Test Description',
        [],
        29.99,
        2
    );

    $book->borrow(); // Заимствуем одну копию
```

```
        $this->expectException(\RuntimeException::class);
        $this->expectExceptionMessage('Cannot archive book with borrowed copies');

        $book->archive();
    }

    public function testToArrayReturnsCorrectStructure(): void
    {
        $book = new Book(
            'Test Book',
            'Test Author',
            2023,
            '978-3-16-148410-0',
            'Fiction',
            300,
            'Test Publisher',
            'English',
            'Test Description',
            ['test', 'book'],
            29.99,
            5
        );

        $array = $book->toArray();

        $this->assertIsArray($array);
        $this->assertArrayHasKey('title', $array);
        $this->assertArrayHasKey('author', $array);
        $this->assertArrayHasKey('isbn', $array);
        $this->assertArrayHasKey('status', $array);
        $this->assertEquals('Test Book', $array['title']);
```

```
    $this->assertEquals(['test', 'book'], $array['keywords']);
}

public function testSetTitleWithEmptyStringThrowsException(): void
{
    $book = new Book(
        'Original Title',
        'Test Author',
        2023,
        '978-3-16-148410-0',
        'Fiction',
        300,
        'Test Publisher',
        'English',
        'Test Description',
        [],
        29.99,
        5
    );

    $this->expectException(\InvalidArgumentException::class);
    $this->expectExceptionMessage('Title cannot be empty');

    $book->setTitle('');
}

public function testSetAvailableCopiesWithInvalidNumberThrowsException(): void
{
    $book = new Book(
        'Test Book',
        'Test Author',
        2023,
```

```
            '978-3-16-148410-0',
            'Fiction',
            300,
            'Test Publisher',
            'English',
            'Test Description',
            [],
            29.99,
            5
        );

        $this->expectException(\InvalidArgumentException::class);

        // Пытаемся установить больше копий, чем всего
        $book->setAvailableCopies(10);
    }
}
```

tests/Unit/Services/CatalogServiceTest.php

php

```
<?php

namespace Tests\Unit\Services;

use PHPUnit\Framework\TestCase;
use SmartLibrary\Services\CatalogService;
use SmartLibrary\Repositories\BookRepository;
use SmartLibrary\Models\Book;

class CatalogServiceTest extends TestCase
{
    private $bookRepositoryMock;
    private $catalogService;
```

```php
protected function setUp(): void
{
    $this->bookRepositoryMock = $this->createMock(BookRepository::class);
    $this->catalogService = new CatalogService($this->bookRepositoryMock);
}

public function testAddBookWithValidData(): void
{
    $expectedBook = $this->createBookStub();

    $this->bookRepositoryMock->expects($this->once())
        ->method('findByIsbn')
        ->with('978-3-16-148410-0')
        ->willReturn(null);

    $this->bookRepositoryMock->expects($this->once())
        ->method('save')
        ->with($this->isInstanceOf(Book::class))
        ->willReturn(1);

    $book = $this->catalogService->addBook(
        'Test Book',
        'Test Author',
        2023,
        '978-3-16-148410-0',
        'Fiction',
        300,
        'Test Publisher',
        'English',
        'Test Description',
        ['test'],
```

```
            29.99,
            5
        );

        $this->assertInstanceOf(Book::class, $book);
    }

    public function testAddBookWithDuplicateIsbnThrowsException(): void
    {
        $existingBook = $this->createBookStub();

        $this->bookRepositoryMock->expects($this->once())
            ->method('findByIsbn')
            ->with('978-3-16-148410-0')
            ->willReturn($existingBook);

        $this->expectException(\RuntimeException::class);
        $this->expectExceptionMessage('Book with this ISBN already exists');

        $this->catalogService->addBook(
            'Test Book',
            'Test Author',
            2023,
            '978-3-16-148410-0',
            'Fiction'
        );
    }

    public function testAddBookWithInvalidCategoryThrowsException(): void
    {
        $this->bookRepositoryMock->expects($this->once())
            ->method('findByIsbn')
```

```
        ->willReturn(null);

    $this->expectException(\InvalidArgumentException::class);
    $this->expectExceptionMessage('Invalid book category');

    $this->catalogService->addBook(
        'Test Book',
        'Test Author',
        2023,
        '978-3-16-148410-0',
        'Invalid Category' // Несуществующая категория
    );
}

public function testFindBooksWithEmptyCriteriaReturnsAllBooks(): void
{
    $expectedBooks = [
        $this->createBookStub(),
        $this->createBookStub()
    ];

    $this->bookRepositoryMock->expects($this->once())
        ->method('findAll')
        ->willReturn($expectedBooks);

    $result = $this->catalogService->findBooks([]);

    $this->assertCount(2, $result);
    $this->assertSame($expectedBooks, $result);
}

public function testFindBooksWithTitleCriteria(): void
```

```
{
    $searchResults = [$this->createBookStub()];

    $this->bookRepositoryMock->expects($this->once())
        ->method('search')
        ->with(['title' => 'Test'])
        ->willReturn($searchResults);

    $result = $this->catalogService->findBooks(['title_like' => 'Test']);

    $this->assertCount(1, $result);
}

public function testGetBookStatisticsReturnsCorrectStructure(): void
{
    $this->bookRepositoryMock->expects($this->once())
        ->method('getTotalBooksCount')
        ->willReturn(10);

    $this->bookRepositoryMock->expects($this->exactly(20)) // 20 категорий
        ->method('countByCategory')
        ->willReturn(1);

    $this->bookRepositoryMock->expects($this->once())
        ->method('findAll')
        ->with(1000)
        ->willReturn([]);

    $statistics = $this->catalogService->getBookStatistics();

    $this->assertIsArray($statistics);
    $this->assertArrayHasKey('total_books', $statistics);
```

```
    $this->assertArrayHasKey('by_category', $statistics);
    $this->assertArrayHasKey('by_year', $statistics);
    $this->assertArrayHasKey('by_availability', $statistics);
    $this->assertEquals(10, $statistics['total_books']);
}

public function testImportBooksReturnsCorrectResults(): void
{
    $booksData = [
        [
            'title' => 'Book 1',
            'author' => 'Author 1',
            'publication_year' => 2023,
            'isbn' => '978-3-16-148410-0',
            'category' => 'Fiction'
        ],
        [
            'title' => 'Book 2',
            'author' => 'Author 2',
            'publication_year' => 2023,
            'isbn' => '978-3-16-148410-1',
            'category' => 'Science'
        ]
    ];

    $this->bookRepositoryMock->expects($this->exactly(2))
        ->method('findByIsbn')
        ->willReturn(null);

    $this->bookRepositoryMock->expects($this->exactly(2))
        ->method('save')
        ->willReturn(1, 2);
```

```
    $result = $this->catalogService->importBooks($booksData);

    $this->assertEquals(2, $result['success']);
    $this->assertEquals(0, $result['failed']);
    $this->assertEmpty($result['errors']);
}

public function testGenerateCategorySummaryReport(): void
{
    // Мокируем getBookStatistics
    $stats = [
        'total_books' => 10,
        'by_category' => ['Fiction' => 5, 'Science' => 5],
        'by_year' => [],
        'by_availability' => []
    ];

    $service = $this->getMockBuilder(CatalogService::class)
        ->setConstructorArgs([$this->bookRepositoryMock])
        ->onlyMethods(['getBookStatistics'])
        ->getMock();

    $service->expects($this->once())
        ->method('getBookStatistics')
        ->willReturn($stats);

    $report = $service->generateReport('category_summary');

    $this->assertEquals('category_summary', $report['type']);
    $this->assertArrayHasKey('Fiction', $report['data']);
    $this->assertArrayHasKey('Science', $report['data']);
```

```
    }

    private function createBookStub(): Book
    {
        return new Book(
            'Test Book',
            'Test Author',
            2023,
            '978-3-16-148410-0',
            'Fiction',
            300,
            'Test Publisher',
            'English',
            'Test Description',
            [],
            29.99,
            5
        );
    }
}
```

tests/Integration/BookRepositoryTest.php

php

```
<?php

namespace Tests\Integration;

use PHPUnit\Framework\TestCase;
use SmartLibrary\Repositories\BookRepository;
use SmartLibrary\Models\Book;
use PDO;

class BookRepositoryTest extends TestCase
```

```
{
    private static $pdo;
    private $repository;

    public static function setUpBeforeClass(): void
    {
        // Создаем соединение с тестовой БД
        self::$pdo = new PDO('sqlite::memory:');
        self::$pdo->setAttribute(PDO::ATTR_ERRMODE,
PDO::ERRMODE_EXCEPTION);

        // Создаем таблицу
        self::$pdo->exec("
            CREATE TABLE books (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                title TEXT NOT NULL,
                author TEXT NOT NULL,
                publication_year INTEGER NOT NULL,
                isbn TEXT NOT NULL UNIQUE,
                category TEXT NOT NULL,
                pages INTEGER NOT NULL,
                publisher TEXT NOT NULL,
                language TEXT NOT NULL,
                description TEXT NOT NULL,
                keywords TEXT NOT NULL,
                price REAL NOT NULL,
                total_copies INTEGER NOT NULL,
                available_copies INTEGER NOT NULL,
                status TEXT NOT NULL DEFAULT 'available',
                created_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
                updated_at TIMESTAMP NULL
            )
```

```
        ");
    }

    protected function setUp(): void
    {
        // Очищаем таблицу перед каждым тестом
        self::$pdo->exec("DELETE FROM books");
        self::$pdo->exec("DELETE FROM sqlite_sequence WHERE name='books'");

        $this->repository = new BookRepository(self::$pdo);
    }

    public function testSaveAndFindBook(): void
    {
        $book = new Book(
            'Integration Test Book',
            'Test Author',
            2023,
            '978-1-23-456789-0',
            'Fiction',
            300,
            'Test Publisher',
            'English',
            'Test Description',
            ['test', 'integration'],
            29.99,
            5
        );

        // Сохраняем книгу
        $id = $this->repository->save($book);
        $this->assertGreaterThan(0, $id);
```

```
    // Ищем книгу по ID
    $foundBook = $this->repository->findById($id);

    $this->assertInstanceOf(Book::class, $foundBook);
    $this->assertEquals($book->getTitle(), $foundBook->getTitle());
    $this->assertEquals($book->getIsbn(), $foundBook->getIsbn());
    $this->assertEquals(5, $foundBook->getAvailableCopies());
}

public function testFindByIsbn(): void
{
    $book = new Book(
        'ISBN Test Book',
        'Test Author',
        2023,
        '978-1-23-456789-1',
        'Science',
        400,
        'Test Publisher',
        'English',
        'Test Description',
        [],
        39.99,
        3
    );

    $this->repository->save($book);

    $foundBook = $this->repository->findByIsbn('978-1-23-456789-1');

    $this->assertNotNull($foundBook);
```

```
    $this->assertEquals('ISBN Test Book', $foundBook->getTitle());
    $this->assertEquals('Science', $foundBook->getCategory());
}

public function testFindAllWithLimit(): void
{
    // Создаем 5 книг
    for ($i = 1; $i <= 5; $i++) {
        $book = new Book(
            "Book $i",
            "Author $i",
            2000 + $i,
            "978-1-23-45678{$i}",
            'Fiction',
            100 + $i * 50,
            'Publisher',
            'English',
            'Description',
            [],
            10.00 + $i,
            $i
        );
        $this->repository->save($book);
    }

    $books = $this->repository->findAll(3);

    $this->assertCount(3, $books);
    $this->assertEquals('Book 1', $books[0]->getTitle());
    $this->assertEquals('Book 2', $books[1]->getTitle());
    $this->assertEquals('Book 3', $books[2]->getTitle());
}
```

```php
public function testUpdateBook(): void
{
    $book = new Book(
        'Original Title',
        'Original Author',
        2020,
        '978-1-23-456789-2',
        'History',
        250,
        'Original Publisher',
        'English',
        'Original Description',
        [],
        19.99,
        2
    );

    $id = $this->repository->save($book);

    // Изменяем книгу
    $book->setTitle('Updated Title');
    $book->setAvailableCopies(1);

    $result = $this->repository->update($book);
    $this->assertTrue($result);

    // Проверяем обновление
    $updatedBook = $this->repository->findById($id);
    $this->assertEquals('Updated Title', $updatedBook->getTitle());
    $this->assertEquals(1, $updatedBook->getAvailableCopies());
}
```

```php
public function testDeleteBook(): void
{
    $book = new Book(
        'Book to Delete',
        'Author',
        2023,
        '978-1-23-456789-3',
        'Fiction',
        300,
        'Publisher',
        'English',
        'Description',
        [],
        29.99,
        1
    );

    $id = $this->repository->save($book);

    // Удаляем книгу
    $result = $this->repository->delete($id);
    $this->assertTrue($result);

    // Проверяем, что книга удалена
    $deletedBook = $this->repository->findById($id);
    $this->assertNull($deletedBook);
}

public function testSearchByTitle(): void
{
    $books = [
```

```
            new Book('PHP Programming', 'Author 1', 2023, '978-111', 'Technology', 400,
'Pub', 'EN', 'Desc', [], 49.99, 3),
            new Book('PHP Cookbook', 'Author 2', 2022, '978-112', 'Technology', 350, 'Pub',
'EN', 'Desc', [], 39.99, 2),
            new Book('JavaScript Guide', 'Author 3', 2023, '978-113', 'Technology', 500, 'Pub',
'EN', 'Desc', [], 59.99, 1),
        ];

        foreach ($books as $book) {
            $this->repository->save($book);
        }

        $results = $this->repository->search(['title' => 'PHP']);

        $this->assertCount(2, $results);
        $this->assertEquals('PHP Programming', $results[0]->getTitle());
        $this->assertEquals('PHP Cookbook', $results[1]->getTitle());
    }

    public function testSearchWithMultipleCriteria(): void
    {
        $books = [
            new Book('Book 1', 'Author A', 2020, '978-121', 'Fiction', 300, 'Pub', 'EN', 'Desc',
[], 29.99, 5),
            new Book('Book 2', 'Author A', 2023, '978-122', 'Fiction', 400, 'Pub', 'EN', 'Desc',
[], 39.99, 0),
            new Book('Book 3', 'Author B', 2022, '978-123', 'Science', 500, 'Pub', 'EN', 'Desc',
[], 49.99, 3),
        ];

        foreach ($books as $book) {
            $this->repository->save($book);
```

```
    }

    $results = $this->repository->search([
        'author' => 'Author A',
        'min_year' => 2021,
        'available_only' => false
    ]);

    $this->assertCount(1, $results);
    $this->assertEquals('Book 2', $results[0]->getTitle());
}

public function testCountByCategory(): void
{
    $categories = ['Fiction', 'Fiction', 'Science', 'History', 'Fiction'];

    foreach ($categories as $i => $category) {
        $book = new Book(
            "Book $i",
            "Author",
            2020 + $i,
            "978-13{$i}",
            $category,
            300,
            'Publisher',
            'English',
            'Description',
            [],
            29.99,
            1
        );
        $this->repository->save($book);
```

```
        }

        $count = $this->repository->countByCategory('Fiction');
        $this->assertEquals(3, $count);
    }

    public function testGetTotalBooksCount(): void
    {
        // Создаем 3 книги
        for ($i = 1; $i <= 3; $i++) {
            $book = new Book(
                "Book $i",
                "Author",
                2023,
                "978-14{$i}",
                'Fiction',
                300,
                'Publisher',
                'English',
                'Description',
                [],
                29.99,
                1
            );
            $this->repository->save($book);
        }

        $count = $this->repository->getTotalBooksCount();
        $this->assertEquals(3, $count);
    }
}
```

tests/Fixtures/TestData.php

```php
<?php

namespace Tests\Fixtures;

use SmartLibrary\Models\Book;

class TestData
{
    public static function getSampleBooks(): array
    {
        return [
            new Book(
                'Clean Code',
                'Robert C. Martin',
                2008,
                '978-0132350884',
                'Technology',
                464,
                'Prentice Hall',
                'English',
                'A handbook of agile software craftsmanship',
                ['programming', 'clean code', 'best practices'],
                39.99,
                10
            ),
            new Book(
                'The Pragmatic Programmer',
                'David Thomas, Andrew Hunt',
                2019,
                '978-0135957059',
                'Technology',
```

```
            352,
            'Addison-Wesley',
            'English',
            'Your journey to mastery',
            ['programming', 'pragmatic', 'software development'],
            44.99,
            8
        ),
        new Book(
            'Design Patterns',
            'Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides',
            1994,
            '978-0201633610',
            'Technology',
            395,
            'Addison-Wesley',
            'English',
            'Elements of Reusable Object-Oriented Software',
            ['design patterns', 'oop', 'software architecture'],
            49.99,
            5
        ),
    ];
}

public static function getInvalidBookData(): array
{
    return [
        [
            'title' => '', // Пустое название
            'author' => 'Author',
            'year' => 2023,
```

```
            'isbn' => '123',
            'category' => 'Fiction'
        ],
        [
            'title' => 'Valid Title',
            'author' => '', // Пустой автор
            'year' => 2023,
            'isbn' => '978-1234567890',
            'category' => 'Fiction'
        ],
        [
            'title' => 'Valid Title',
            'author' => 'Author',
            'year' => 500, // Неверный год
            'isbn' => '978-1234567890',
            'category' => 'Fiction'
        ]
    ];
}

public static function getSearchCriteria(): array
{
    return [
        'simple_title' => ['title' => 'Code'],
        'author_search' => ['author' => 'Martin'],
        'year_range' => ['min_year' => 2000, 'max_year' => 2010],
        'available_only' => ['available_only' => true],
        'complex' => [
            'title' => 'Pattern',
            'category' => 'Technology',
            'min_year' => 1990,
            'available_only' => true
```

```
            ]
        ];
    }
}
```

tests/bootstrap.php

php

```php
<?php

require_once __DIR__ . '/../vendor/autoload.php';

// Настройка тестового окружения
if (!defined('TEST_MODE')) {
    define('TEST_MODE', true);
}

// Установка временной зоны для тестов
date_default_timezone_set('UTC');
```

phpunit.xml

xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<phpunit xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:noNamespaceSchemaLocation="https://schema.phpunit.de/10.5/phpunit.xsd"
         bootstrap="tests/bootstrap.php"
         colors="true"
         cacheDirectory=".phpunit.cache">
    <testsuites>
        <testsuite name="Unit Tests">
            <directory>tests/Unit</directory>
        </testsuite>
        <testsuite name="Integration Tests">
            <directory>tests/Integration</directory>
        </testsuite>
```

```
    </testsuites>

    <source>
        <include>
            <directory>src</directory>
        </include>
        <exclude>
            <directory>vendor</directory>
            <directory>tests</directory>
            <directory>reports</directory>
        </exclude>
    </source>

    <coverage>
        <report>
            <html outputDirectory="reports/coverage"/>
            <text outputFile="reports/coverage.txt"/>
            <clover outputFile="reports/coverage.xml"/>
        </report>
    </coverage>

    <php>
        <ini name="error_reporting" value="-1"/>
        <ini name="display_errors" value="1"/>
        <ini name="display_startup_errors" value="1"/>
        <server name="APP_ENV" value="test"/>
    </php>
</phpunit>
```

composer.json

json

```
{
    "name": "smart-library/library-system",
```

```
    "description": "Smart Library System with comprehensive testing",
    "type": "project",
    "require": {
        "php": ">=8.1"
    },
    "require-dev": {
        "phpunit/phpunit": "^10.0",
        "mockery/mockery": "^1.5",
        "squizlabs/php_codesniffer": "^3.7",
        "phpmd/phpmd": "^2.13",
        "phpmetrics/phpmetrics": "^2.8",
        "vimeo/psalm": "^5.15",
        "phpstan/phpstan": "^1.10"
    },
    "autoload": {
        "psr-4": {
            "SmartLibrary\\": "src/"
        }
    },
    "autoload-dev": {
        "psr-4": {
            "Tests\\": "tests/"
        }
    },
    "scripts": {
        "test": "phpunit",
        "test-coverage": "phpunit --coverage-html reports/coverage",
        "check-style": "phpcs --standard=PSR12 src tests",
        "fix-style": "phpcbf --standard=PSR12 src tests",
        "phpmd": "phpmd src text
cleancode,codesize,controversial,design,naming,unusedcode",
        "phpmetrics": "phpmetrics --report-html=reports/metrics src",
```

```
        "static-analysis": "phpstan analyse src --level=8",
        "psalm": "psalm",
        "ci": [
            "@check-style",
            "@static-analysis",
            "@test"
        ]
    },
    "config": {
        "sort-packages": true
    }
}
```

.gitlab-ci.yml (пример конфигурации CI/CD)

```
yaml
image: php:8.1

stages:
  - test
  - analysis
  - deploy

variables:
  COMPOSER_CACHE_DIR: "${CI_PROJECT_DIR}/.composer-cache"

cache:
  paths:
    - vendor/
    - .composer-cache/

before_script:
  - apt-get update -yqq
  - apt-get install -yqq git libsqlite3-dev sqlite3
```

```
  - docker-php-ext-install pdo pdo_sqlite
  - curl -sS https://getcomposer.org/installer | php
  - php composer.phar install --prefer-dist --no-progress --no-suggest

unit-tests:
  stage: test
  script:
    - php vendor/bin/phpunit --testsuite "Unit Tests" --coverage-text --colors=never

integration-tests:
  stage: test
  script:
    - php vendor/bin/phpunit --testsuite "Integration Tests"

coverage:
  stage: test
  script:
    - php vendor/bin/phpunit --coverage-html reports/coverage
  artifacts:
    paths:
      - reports/coverage/
    expire_in: 1 week

code-style:
  stage: analysis
  script:
    - php vendor/bin/phpcs --standard=PSR12 --colors src tests

phpmd:
  stage: analysis
  script:
```

```
    - php vendor/bin/phpmd src text
cleancode,codesize,controversial,design,naming,unusedcode --reportfile
reports/phpmd/report.txt
  artifacts:
    paths:
      - reports/phpmd/
    expire_in: 1 week

phpmetrics:
  stage: analysis
  script:
    - php vendor/bin/phpmetrics --report-html=reports/metrics src
  artifacts:
    paths:
      - reports/metrics/
    expire_in: 1 week

phpstan:
  stage: analysis
  script:
    - php vendor/bin/phpstan analyse src --level=8 --no-progress

pages:
  stage: deploy
  script:
    - mkdir public
    - cp -r reports/coverage public/
    - cp -r reports/metrics public/
  artifacts:
    paths:
      - public
  only:
```

- master

3. Скрипты для анализа метрик

scripts/analyze-metrics.php

php

```php
<?php

/**
 * Скрипт для анализа метрик кода
 */

require_once __DIR__ . '/../vendor/autoload.php';

echo "=== Анализ метрик кода CatalogService ===\n\n";

// 1. Ручной расчет метрик для метода findBooks()
echo "1. РУЧНОЙ РАСЧЕТ МЕТРИК:\n";
echo "   Метод: CatalogService::findBooks()\n";

// Цикломатическая сложность (оценка)
echo "   - Цикломатическая сложность (V(G)):\n";
echo "     Основные if условия: 8\n";
echo "     Вложенные условия в циклах: +3\n";
echo "     Логические операторы: +2\n";
echo "     Итого V(G) = 8 + 3 + 2 + 1 = 14\n\n";

// Количество строк кода
echo "   - Количество строк кода (SLOC): ~120 строк\n";
echo "     (без комментариев и пустых строк)\n\n";

// Индекс поддерживаемости
echo "   - Индекс поддерживаемости (MI):\n";
echo "     MI = 171 - 5.2 * ln(V) - 0.23 * ln(SLOC) - 16.2 * ln(LOC)\n";
```

```
echo "    Оценка: 65-75 (умеренная поддерживаемость)\n\n";

// 2. Запуск инструментов анализа
echo "2. АВТОМАТИЧЕСКИЙ АНАЛИЗ ИНСТРУМЕНТАМИ:\n";

// PHPMD
echo "   Запуск PHPMD...\n";
exec('vendor/bin/phpmd src/Services/CatalogService.php text
cleancode,codesize,controversial,design,naming,unusedcode', $phpmdOutput);
foreach ($phpmdOutput as $line) {
    echo "   $line\n";
}

// PHP Metrics
echo "\n   Генерация отчета PHP Metrics...\n";
exec('vendor/bin/phpmetrics --report-json=reports/metrics/metrics.json src/Services/');

// 3. Предложения по рефакторингу
echo "\n3. ПРЕДЛОЖЕНИЯ ПО РЕФАКТОРИНГУ:\n";
echo "   ПРОБЛЕМА 1: Слишком длинный метод findBooks()\n";
echo "   Решение: Разделить на подметоды:\n";
echo "     - searchByTitle()\n";
echo "     - searchByAuthor()\n";
echo "     - searchByCategory()\n";
echo "     - filterByAvailability()\n";
echo "     - sortResults()\n\n";

echo "   ПРОБЛЕМА 2: Высокая связанность с BookRepository\n";
echo "   Решение: Внедрить интерфейс и использовать Dependency Injection\n\n";

echo "   ПРОБЛЕМА 3: Много параметров в addBook()\n";
echo "   Решение: Использовать DTO (Data Transfer Object)\n";
```

```
echo "    class BookData {\n";
echo "        public string \$title;\n";
echo "        public string \$author;\n";
echo "        // ... остальные поля\n";
echo "    }\n\n";

echo "  ПРОБЛЕМА 4: Дублирование валидации\n";
echo "  Решение: Вынести валидацию в отдельный класс Validator\n\n";

echo "  ПРОБЛЕМА 5: Жестко закодированные категории\n";
echo "  Решение: Хранить категории в базе данных или конфигурации\n";

// 4. Пример рефакторинга
echo "\n4. ПРИМЕР РЕФАКТОРИНГА:\n";
echo <<<'EXAMPLE'
  БЫЛО:
  public function findBooks(array $criteria): array {
      // 120 строк сложного кода
  }

  СТАЛО:
  public function findBooks(array $criteria): array {
      $results = $this->searchBooks($criteria);
      $results = $this->filterBooks($results, $criteria);
      $results = $this->sortBooks($results, $criteria);
      return $this->limitResults($results, $criteria);
  }

  private function searchBooks(array $criteria): array {
      // Поиск по отдельным критериям
  }
```

```
    private function filterBooks(array $books, array $criteria): array {
        // Фильтрация результатов
    }

    private function sortBooks(array $books, array $criteria): array {
        // Сортировка
    }

    private function limitResults(array $books, array $criteria): array {
        // Ограничение количества
    }
EXAMPLE;

echo "\n\n=== Анализ завершен ===\n";
```

4. Инструкция по использованию.

Установка и запуск тестов:

```
bash
# 1. Клонировать проект
git clone https://github.com/your-username/smart-library.git
cd smart-library

# 2. Установить зависимости
composer install

# 3. Запустить все тесты
composer test

# 4. Запустить тесты с покрытием
composer test-coverage

# 5. Проверить стиль кода
composer check-style
```

```
# 6. Исправить стиль автоматически
composer fix-style

# 7. Запустить анализ метрик
composer phpmd
composer phpmetrics

# 8. Статический анализ
composer static-analysis
composer psalm

# 9. Запустить полный CI-пайплайн
composer ci
```

Структура отчётов после выполнения:

```
text
reports/
├── coverage/          # Отчёт о покрытии тестами
├── metrics/           # Отчёт PHP Metrics
└── phpmd/             # Отчёт PHPMD
```

Python:

Python-модуль "Book Similarity Analyzer" для интеграции с PHP-проектом

Структура модуля

```
text
book-similarity/
├── similarity/
│   ├── __init__.py
│   ├── analyzer.py          # Основной класс анализа схожести
│   ├── metrics_calculator.py # Калькулятор метрик с проблемами
│   ├── text_processor.py     # Обработка текста
│   └── similarity_algorithms.py # Алгоритмы сравнения
│
```

```
├── tests/
│   ├── __init__.py
│   ├── test_analyzer.py
│   ├── test_metrics_calculator.py
│   └── test_integration.py
│
├── examples/
│   ├── php_integration.php
│   └── python_usage.py
│
├── requirements.txt
├── setup.py
└── README.md
```

1. Основной код модуля (с намеренными проблемами для анализа)

similarity/analyzer.py

python

```
"""
Модуль анализа схожести книг с намеренными проблемами для анализа метриками
"""

import re
import math
import string
import hashlib
from collections import Counter, defaultdict
from typing import Dict, List, Tuple, Optional, Union, Any, Set
from dataclasses import dataclass
import json
import sys


@dataclass
```

```python
class BookData:
    """DTO для данных книги"""
    id: int
    title: str
    author: str
    description: str = ""
    keywords: List[str] = None
    category: str = ""
    year: int = 0
    isbn: str = ""

    def __post_init__(self):
        if self.keywords is None:
            self.keywords = []


class BookSimilarityAnalyzer:
    """
    Анализатор схожести книг - специально с проблемами для анализа метрик
    """

    def __init__(self, config: Optional[Dict] = None):
        self.config = config or {}
        self._cache = {}
        self._stats = {
            'comparisons': 0,
            'cache_hits': 0,
            'processing_time': 0
        }
        self._stop_words = self._load_stop_words()
        self._similarity_threshold = self.config.get('threshold', 0.7)
```

```
    def _load_stop_words(self) -> Set[str]:
        """Загрузка стоп-слов (жестко закодировано для примера)"""
        return {
            'и', 'в', 'во', 'не', 'что', 'он', 'на', 'я', 'с', 'со', 'как', 'а', 'то', 'все', 'она',
            'так', 'его', 'но', 'да', 'ты', 'к', 'у', 'же', 'вы', 'за', 'бы', 'по', 'только', 'ее',
            'мне', 'было', 'вот', 'от', 'меня', 'еще', 'нет', 'о', 'из', 'ему', 'теперь', 'когда',
            'даже', 'ну', 'вдруг', 'ли', 'если', 'уже', 'или', 'ни', 'быть', 'был', 'него', 'до',
            'вас', 'нибудь', 'опять', 'уж', 'вам', 'ведь', 'там', 'потом', 'себя', 'ничего', 'ей',
            'может', 'они', 'тут', 'где', 'есть', 'надо', 'ней', 'для', 'мы', 'тебя', 'их', 'чем',
            'была', 'сам', 'чтоб', 'без', 'будто', 'чего', 'раз', 'тоже', 'себе', 'под', 'будет',
            'ж', 'тогда', 'кто', 'этот', 'того', 'потому', 'этого', 'какой', 'совсем', 'ним',
            'здесь', 'этом', 'один', 'почти', 'мой', 'тем', 'чтобы', 'нее', 'сейчас', 'были', 'куда',
            'зачем', 'всех', 'никогда', 'можно', 'при', 'наконец', 'два', 'об', 'другой', 'хоть',
            'после', 'над', 'больше', 'тот', 'через', 'эти', 'нас', 'про', 'всего', 'них', 'какая',
            'много', 'разве', 'три', 'эту', 'моя', 'впрочем', 'хорошо', 'свою', 'этой', 'перед',
            'иногда', 'лучше', 'чуть', 'том', 'нельзя', 'такой', 'им', 'более', 'всегда', 'конечно',
            'всю', 'между', 'the', 'and', 'or', 'but', 'in', 'on', 'at', 'to', 'for', 'of', 'with',
            'by', 'from', 'up', 'about', 'into', 'over', 'after', 'a', 'и', 'но', 'или', 'если'
        }

    def calculate_similarity(self, book1: BookData, book2: BookData,
                             method: str = "combined") -> Dict[str, float]:
        """
        Основной метод расчета схожести с высокой цикломатической сложностью

        Args:
            book1: Первая книга
            book2: Вторая книга
            method: Метод расчета (combined, title, author, content, jaccard, cosine, levenshtein)

        Returns:
```

```
        Словарь с результатами схожести по разным метрикам
    """
    cache_key = f"{book1.id}_{book2.id}_{method}"
    if cache_key in self._cache:
        self._stats['cache_hits'] += 1
        return self._cache[cache_key]

    self._stats['comparisons'] += 1

    results = {}

    if method == "combined" or method == "title":
        results['title_similarity'] = self._calculate_title_similarity(book1.title, book2.title)

    if method == "combined" or method == "author":
        results['author_similarity']     =     self._calculate_author_similarity(book1.author,
book2.author)

    if method == "combined" or method == "content":
        if book1.description and book2.description:
            results['content_similarity'] = self._calculate_content_similarity(
                book1.description, book2.description
            )
        else:
            results['content_similarity'] = 0.0

    if method == "combined" or method == "jaccard":
        results['jaccard_similarity'] = self._calculate_jaccard_similarity(
            book1.keywords, book2.keywords
        )

    if method == "combined" or method == "cosine":
```

```
        results['cosine_similarity'] = self._calculate_cosine_similarity(
            book1.description, book2.description
        )

    if method == "combined" or method == "levenshtein":
        results['levenshtein_similarity'] = self._calculate_levenshtein_similarity(
            book1.title, book2.title
        )

    if method == "combined":
        # Комбинированная оценка с весами
        weights = {
            'title': 0.3,
            'author': 0.25,
            'content': 0.2,
            'jaccard': 0.15,
            'cosine': 0.05,
            'levenshtein': 0.05
        }

        combined_score = 0.0
        weight_sum = 0.0

        for metric, weight in weights.items():
            metric_key = f"{metric}_similarity"
            if metric_key in results and results[metric_key] is not None:
                if metric == 'levenshtein':
                    # Инвертируем расстояние Левенштейна
                    score = 1.0 - results[metric_key]
                else:
                    score = results[metric_key]
```

```
                combined_score += score * weight
                weight_sum += weight

        if weight_sum > 0:
            results['combined_similarity'] = combined_score / weight_sum
        else:
            results['combined_similarity'] = 0.0

        # Добавляем категоричную оценку
        if results['combined_similarity'] > 0.9:
            results['similarity_level'] = 'VERY_HIGH'
        elif results['combined_similarity'] > 0.7:
            results['similarity_level'] = 'HIGH'
        elif results['combined_similarity'] > 0.5:
            results['similarity_level'] = 'MEDIUM'
        elif results['combined_similarity'] > 0.3:
            results['similarity_level'] = 'LOW'
        else:
            results['similarity_level'] = 'VERY_LOW'

    self._cache[cache_key] = results
    return results

def _calculate_title_similarity(self, title1: str, title2: str) -> float:
    """Сложный метод сравнения названий"""
    if not title1 or not title2:
        return 0.0

    # Нормализация
    t1 = self._normalize_text(title1.lower())
    t2 = self._normalize_text(title2.lower())
```

```
if t1 == t2:
    return 1.0

# Разбиваем на слова
words1 = set(t1.split())
words2 = set(t2.split())

if not words1 or not words2:
    return 0.0

# Удаляем стоп-слова
words1 = {w for w in words1 if w not in self._stop_words}
words2 = {w for w in words2 if w not in self._stop_words}

if not words1 or not words2:
    return 0.0

# Коэффициент Жаккара
intersection = len(words1.intersection(words2))
union = len(words1.union(words2))

if union == 0:
    return 0.0

jaccard = intersection / union

# Дополнительные проверки
if len(words1) == 1 and len(words2) == 1:
    # Одно слово в каждом названии
    word1 = list(words1)[0]
    word2 = list(words2)[0]
```

```
        if word1 in word2 or word2 in word1:
            return max(jaccard, 0.8)

    # Проверка на наличие чисел
    numbers1 = set(re.findall(r'\d+', title1))
    numbers2 = set(re.findall(r'\d+', title2))

    if numbers1 and numbers2:
        if numbers1 != numbers2:
            jaccard *= 0.7  # Штраф за разные числа

    # Длина названий
    len1 = len(title1)
    len2 = len(title2)
    len_ratio = min(len1, len2) / max(len1, len2) if max(len1, len2) > 0 else 1.0

    # Средний коэффициент
    similarity = (jaccard * 0.6) + (len_ratio * 0.4)

    return min(similarity, 1.0)

def _calculate_author_similarity(self, author1: str, author2: str) -> float:
    """Сложный метод сравнения авторов"""
    if not author1 or not author2:
        return 0.0

    a1 = self._normalize_text(author1.lower().strip())
    a2 = self._normalize_text(author2.lower().strip())

    if a1 == a2:
        return 1.0
```

```
    # Разбиваем на части (имя, фамилия, отчество)
    parts1 = a1.split()
    parts2 = a2.split()

    if not parts1 or not parts2:
        return 0.0

    # Проверка на инициалы
    if len(parts1) == 1 and len(parts2) > 1:
        # a1 может быть фамилией, a2 - полным именем
        if parts1[0] == parts2[-1]:  # Сравниваем фамилии
            return 0.8

    if len(parts2) == 1 and len(parts1) > 1:
        if parts2[0] == parts1[-1]:
            return 0.8

    # Коэффициент Жаккара для множеств частей
    set1 = set(parts1)
    set2 = set(parts2)

    intersection = len(set1.intersection(set2))
    union = len(set1.union(set2))

    if union == 0:
        return 0.0

    jaccard = intersection / union

    # Проверка на общие подстроки
    common_chars = 0
    for p1 in parts1:
```

```
            for p2 in parts2:
                if p1 in p2 or p2 in p1:
                    common_chars += 1

        substring_factor = common_chars / max(len(parts1), len(parts2))

        # Комбинированный результат
        similarity = (jaccard * 0.7) + (substring_factor * 0.3)

        return min(similarity, 1.0)

    def _calculate_content_similarity(self, text1: str, text2: str) -> float:
        """Сложный метод сравнения содержимого"""
        if not text1 or not text2:
            return 0.0

        # Нормализация
        t1 = self._normalize_text(text1.lower())
        t2 = self._normalize_text(text2.lower())

        if t1 == t2:
            return 1.0

        # Разбиваем на слова
        words1 = t1.split()
        words2 = t2.split()

        if not words1 or not words2:
            return 0.0

        # Удаляем стоп-слова
        words1 = [w for w in words1 if w not in self._stop_words]
```

```
words2 = [w for w in words2 if w not in self._stop_words]

if not words1 or not words2:
    return 0.0

# Создаем векторы слов
vector1 = Counter(words1)
vector2 = Counter(words2)

# Все уникальные слова
all_words = set(vector1.keys()).union(set(vector2.keys()))

if not all_words:
    return 0.0

# Косинусная схожесть
dot_product = 0.0
norm1 = 0.0
norm2 = 0.0

for word in all_words:
    v1 = vector1.get(word, 0)
    v2 = vector2.get(word, 0)

    dot_product += v1 * v2
    norm1 += v1 ** 2
    norm2 += v2 ** 2

if norm1 == 0 or norm2 == 0:
    return 0.0

cosine = dot_product / (math.sqrt(norm1) * math.sqrt(norm2))
```

```
        # Дополнительные метрики
        words_set1 = set(words1)
        words_set2 = set(words2)

        intersection = len(words_set1.intersection(words_set2))
        union = len(words_set1.union(words_set2))

        if union == 0:
            jaccard = 0.0
        else:
            jaccard = intersection / union

        # Длина текстов
        len1 = len(text1)
        len2 = len(text2)
        len_ratio = min(len1, len2) / max(len1, len2) if max(len1, len2) > 0 else 1.0

        # Комбинированная оценка
        similarity = (cosine * 0.5) + (jaccard * 0.3) + (len_ratio * 0.2)

        return min(similarity, 1.0)

    def _calculate_jaccard_similarity(self, list1: List[str], list2: List[str]) -> float:
        """Коэффициент Жаккара для списков ключевых слов"""
        if not list1 or not list2:
            return 0.0

        set1 = set(self._normalize_text(kw.lower()) for kw in list1)
        set2 = set(self._normalize_text(kw.lower()) for kw in list2)

        intersection = len(set1.intersection(set2))
```

```
        union = len(set1.union(set2))

        if union == 0:
            return 0.0

        return intersection / union

    def _calculate_cosine_similarity(self, text1: str, text2: str) -> float:
        """Косинусная схожесть с TF-IDF (упрощенная версия)"""
        if not text1 or not text2:
            return 0.0

        # Нормализация
        t1 = self._normalize_text(text1.lower())
        t2 = self._normalize_text(text2.lower())

        if t1 == t2:
            return 1.0

        # Разбиваем на слова
        words1 = t1.split()
        words2 = t2.split()

        if not words1 or not words2:
            return 0.0

        # Удаляем стоп-слова
        words1 = [w for w in words1 if w not in self._stop_words]
        words2 = [w for w in words2 if w not in self._stop_words]

        if not words1 or not words2:
            return 0.0
```

```
        # Все уникальные слова
        all_words = list(set(words1 + words2))

        # Векторы частот
        vector1 = [words1.count(word) for word in all_words]
        vector2 = [words2.count(word) for word in all_words]

        # Косинусная схожесть
        dot_product = sum(v1 * v2 for v1, v2 in zip(vector1, vector2))
        norm1 = math.sqrt(sum(v ** 2 for v in vector1))
        norm2 = math.sqrt(sum(v ** 2 for v in vector2))

        if norm1 == 0 or norm2 == 0:
            return 0.0

        return dot_product / (norm1 * norm2)

    def _calculate_levenshtein_similarity(self, str1: str, str2: str) -> float:
        """Расстояние Левенштейна (нормализованное)"""
        if not str1 or not str2:
            return 0.0

        # Приводим к нижнему регистру
        s1 = str1.lower()
        s2 = str2.lower()

        if s1 == s2:
            return 0.0  # Расстояние 0

        len1 = len(s1)
        len2 = len(s2)
```

```
    # Создаем матрицу
    matrix = [[0] * (len2 + 1) for _ in range(len1 + 1)]

    for i in range(len1 + 1):
        matrix[i][0] = i
    for j in range(len2 + 1):
        matrix[0][j] = j

    # Заполняем матрицу
    for i in range(1, len1 + 1):
        for j in range(1, len2 + 1):
            cost = 0 if s1[i-1] == s2[j-1] else 1
            matrix[i][j] = min(
                matrix[i-1][j] + 1,      # Удаление
                matrix[i][j-1] + 1,      # Вставка
                matrix[i-1][j-1] + cost  # Замена
            )

    distance = matrix[len1][len2]
    max_len = max(len1, len2)

    if max_len == 0:
        return 0.0

    return distance / max_len

def _normalize_text(self, text: str) -> str:
    """Нормализация текста"""
    if not text:
        return ""
```

```
    # Удаляем пунктуацию
    text = text.translate(str.maketrans('', '', string.punctuation))

    # Заменяем множественные пробелы на один
    text = re.sub(r'\s+', ' ', text)

    return text.strip()

def find_similar_books(self, target_book: BookData,
                       book_list: List[BookData],
                       threshold: float = 0.7,
                       max_results: int = 10) -> List[Tuple[BookData, Dict[str, float]]]:
    """
    Поиск похожих книг - сложный метод с вложенными циклами и условиями
    """
    if not book_list:
        return []

    results = []

    for book in book_list:
        if book.id == target_book.id:
            continue

        # Пропускаем книги без необходимых данных
        if not book.title or not book.author:
            continue

        # Быстрая проверка по автору
        author_sim = self._calculate_author_similarity(target_book.author, book.author)
        if author_sim < 0.3:  # Если авторы совсем разные, пропускаем
            continue
```

```
            # Полный расчет схожести
            similarity = self.calculate_similarity(target_book, book, "combined")

            combined_score = similarity.get('combined_similarity', 0.0)

            if combined_score >= threshold:
                results.append((book, similarity))

        # Сортировка по убыванию схожести
        results.sort(key=lambda x: x[1].get('combined_similarity', 0), reverse=True)

        # Ограничение количества результатов
        if max_results > 0 and len(results) > max_results:
            results = results[:max_results]

        return results

    def get_recommendations(self, user_books: List[BookData],
                            all_books: List[BookData],
                            user_preferences: Optional[Dict] = None) -> List[BookData]:
        """
        Генерация рекомендаций на основе схожести
        """
        if not user_books or not all_books:
            return []

        preferences = user_preferences or {}

        # Собираем оценки схожести для всех книг пользователя
        book_scores = defaultdict(float)
        book_counts = defaultdict(int)
```

```
for user_book in user_books:
    similar_books = self.find_similar_books(
        user_book, all_books,
        threshold=preferences.get('similarity_threshold', 0.6),
        max_results=20
    )

    for similar_book, similarity_data in similar_books:
        score = similarity_data.get('combined_similarity', 0)

        # Учитываем предпочтения пользователя
        if preferences.get('prefer_same_author', False):
            author_sim = similarity_data.get('author_similarity', 0)
            score *= (1.0 + author_sim * 0.5)

        if preferences.get('prefer_same_category', False) and user_book.category:
            if user_book.category == similar_book.category:
                score *= 1.3

        # Учитываем год издания
        if preferences.get('prefer_recent', False) and similar_book.year:
            current_year = 2024  # Можно получить из datetime
            age = current_year - similar_book.year
            if age <= 5:  # Книги не старше 5 лет
                score *= 1.2

        book_scores[similar_book.id] += score
        book_counts[similar_book.id] += 1

# Убираем книги, которые уже есть у пользователя
user_book_ids = {book.id for book in user_books}
```

```
        # Вычисляем средние оценки
        recommendations = []
        for book in all_books:
            if book.id in user_book_ids:
                continue

            if book.id in book_scores and book_counts[book.id] > 0:
                avg_score = book_scores[book.id] / book_counts[book.id]

                # Порог рекомендации
                if avg_score >= preferences.get('recommendation_threshold', 0.5):
                    recommendations.append((book, avg_score))

        # Сортировка по убыванию оценки
        recommendations.sort(key=lambda x: x[1], reverse=True)

        # Ограничение количества рекомендаций
        max_recs = preferences.get('max_recommendations', 10)
        if max_recs > 0 and len(recommendations) > max_recs:
            recommendations = recommendations[:max_recs]

        return [book for book, _ in recommendations]

    def get_statistics(self) -> Dict[str, Any]:
        """Получение статистики работы анализатора"""
        return {
            'total_comparisons': self._stats['comparisons'],
            'cache_hits': self._stats['cache_hits'],
            'cache_size': len(self._cache),
            'cache_hit_rate': (self._stats['cache_hits'] / self._stats['comparisons'])
                              if self._stats['comparisons'] > 0 else 0
```

```
    }
```

similarity/metrics_calculator.py

```
python
"""
Калькулятор метрик качества для анализа кода - намеренно сложный
"""

import math
import statistics
from typing import List, Dict, Any, Tuple, Optional
from collections import defaultdict


class CodeMetricsCalculator:
    """
    Класс для расчета метрик качества кода
    Специально содержит сложный код для анализа
    """

    def __init__(self):
        self.metrics_cache = {}
        self.history = []

    def calculate_all_metrics(self, code_data: Dict[str, Any]) -> Dict[str, float]:
        """
        Основной метод расчета всех метрик - высокая цикломатическая сложность
        """
        if not code_data:
            return {}

        results = {}
```

```
        # Метрики размера
        if 'lines' in code_data:
            results['total_lines'] = code_data['lines']

            if 'comments' in code_data:
                results['comment_density'] = (
                    code_data['comments'] / code_data['lines'] * 100
                    if code_data['lines'] > 0 else 0
                )

        # Метрики сложности
        if 'functions' in code_data:
            results['function_count'] = len(code_data['functions'])

            complexities = []
            for func in code_data['functions']:
                if 'complexity' in func:
                    complexities.append(func['complexity'])

            if complexities:
                results['avg_complexity'] = statistics.mean(complexities)
                results['max_complexity'] = max(complexities)
                results['min_complexity'] = min(complexities)
                results['complexity_std'] = statistics.stdev(complexities) if len(complexities) > 1
else 0

                # Оценка качества по сложности
                if results['avg_complexity'] < 5:
                    results['complexity_rating'] = 'EXCELLENT'
                elif results['avg_complexity'] < 10:
                    results['complexity_rating'] = 'GOOD'
                elif results['avg_complexity'] < 20:
```

```
            results['complexity_rating'] = 'FAIR'
        else:
            results['complexity_rating'] = 'POOR'

    # Метрики связности
    if 'classes' in code_data:
        results['class_count'] = len(code_data['classes'])

        methods_per_class = []
        for cls in code_data['classes']:
            if 'methods' in cls:
                methods_per_class.append(len(cls['methods']))

        if methods_per_class:
            results['avg_methods_per_class'] = statistics.mean(methods_per_class)
            results['max_methods_per_class'] = max(methods_per_class)

            if results['avg_methods_per_class'] < 5:
                results['cohesion_rating'] = 'HIGH'
            elif results['avg_methods_per_class'] < 10:
                results['cohesion_rating'] = 'MEDIUM'
            else:
                results['cohesion_rating'] = 'LOW'

    # Метрики наследования
    if 'inheritance_depth' in code_data:
        depth = code_data['inheritance_depth']
        results['inheritance_depth'] = depth

        if depth == 0:
            results['inheritance_rating'] = 'NO_INHERITANCE'
        elif depth == 1:
```

```
            results['inheritance_rating'] = 'SHALLOW'
        elif depth == 2:
            results['inheritance_rating'] = 'MODERATE'
        else:
            results['inheritance_rating'] = 'DEEP'

    # Метрики связанности
    if 'dependencies' in code_data:
        deps = code_data['dependencies']
        results['dependency_count'] = len(deps)

        if results['dependency_count'] < 3:
            results['coupling_rating'] = 'LOW'
        elif results['dependency_count'] < 7:
            results['coupling_rating'] = 'MEDIUM'
        else:
            results['coupling_rating'] = 'HIGH'

    # Индекс поддерживаемости
    if all(key in results for key in ['avg_complexity', 'total_lines', 'dependency_count']):
        mi = self._calculate_maintainability_index(
            results['avg_complexity'],
            results['total_lines'],
            results['dependency_count'],
            results.get('comment_density', 0)
        )
        results['maintainability_index'] = mi

        if mi > 85:
            results['maintainability_rating'] = 'EXCELLENT'
        elif mi > 65:
            results['maintainability_rating'] = 'GOOD'
```

```
        elif mi > 45:
            results['maintainability_rating'] = 'FAIR'
        else:
            results['maintainability_rating'] = 'POOR'

    # Общая оценка
    if all(rating in results for rating in
           ['complexity_rating', 'cohesion_rating', 'coupling_rating',
'maintainability_rating']):

        rating_scores = {
            'EXCELLENT': 4,
            'GOOD': 3,
            'FAIR': 2,
            'POOR': 1,
            'HIGH': 4,
            'MEDIUM': 2,
            'LOW': 1,
            'NO_INHERITANCE': 3,
            'SHALLOW': 4,
            'MODERATE': 3,
            'DEEP': 1
        }

        total_score = 0
        weights = {
            'complexity_rating': 0.3,
            'cohesion_rating': 0.2,
            'coupling_rating': 0.2,
            'maintainability_rating': 0.3
        }
```

```
        for rating_key, weight in weights.items():
            rating = results[rating_key]
            if rating in rating_scores:
                total_score += rating_scores[rating] * weight

        results['overall_score'] = total_score

        if total_score > 3.5:
            results['overall_rating'] = 'EXCELLENT'
        elif total_score > 2.5:
            results['overall_rating'] = 'GOOD'
        elif total_score > 1.5:
            results['overall_rating'] = 'FAIR'
        else:
            results['overall_rating'] = 'POOR'

    # Сохраняем в историю
    self.history.append({
        'timestamp': '2024-01-15',  # Можно использовать datetime
        'metrics': results
    })

    return results

def _calculate_maintainability_index(self, complexity: float, lines: int,
                                     dependencies: int, comment_density: float) -> float:
    """Расчет индекса поддерживаемости"""
    if complexity <= 0 or lines <= 0:
        return 100.0

    # Формула на основе Halstead Volume и цикломатической сложности
    halstead_volume = lines * math.log2(lines + 1) if lines > 0 else 0
```

```
        mi = 171.0
        mi -= 5.2 * math.log(complexity + 1)
        mi -= 0.23 * math.log(lines + 1)
        mi -= 16.2 * math.log(halstead_volume + 1)
        mi += 50 * math.sin(math.sqrt(2.4 * comment_density))

        # Корректировка на зависимости
        mi -= dependencies * 0.1

        # Ограничиваем значение
        return max(0, min(100, mi))

    def analyze_trends(self, metric_name: str, window_size: int = 5) -> Dict[str, Any]:
        """Анализ трендов метрик"""
        if len(self.history) < 2:
            return {'trend': 'INSUFFICIENT_DATA'}

        metrics_data = []
        for entry in self.history[-window_size:]:
            if metric_name in entry['metrics']:
                metrics_data.append(entry['metrics'][metric_name])

        if len(metrics_data) < 2:
            return {'trend': 'INSUFFICIENT_DATA'}

        # Расчет тренда
        from scipy import stats

        try:
            x = list(range(len(metrics_data)))
            slope, intercept, r_value, p_value, std_err = stats.linregress(x, metrics_data)
```

```
            trend = 'STABLE'
            if slope > 0.1:
                trend = 'IMPROVING'
            elif slope < -0.1:
                trend = 'DETERIORATING'

            return {
                'trend': trend,
                'slope': slope,
                'r_squared': r_value ** 2,
                'current_value': metrics_data[-1],
                'avg_value': statistics.mean(metrics_data)
            }
        except:
            return {'trend': 'ANALYSIS_ERROR'}
```

similarity/text_processor.py

python

```
"""
Обработчик текста для анализа
"""

import re
from typing import List, Set


class TextProcessor:
    """Класс для обработки текста"""

    def __init__(self):
        self._stop_words = self._load_stop_words()
```

```
def _load_stop_words(self) -> Set[str]:
    """Загрузка стоп-слов"""
    # Упрощенный список для примера
    return {
        'и', 'в', 'во', 'не', 'что', 'он', 'на', 'я', 'с', 'со', 'как', 'а', 'то', 'все',
        'она', 'так', 'его', 'но', 'да', 'ты', 'к', 'у', 'же', 'вы', 'за', 'бы', 'по',
        'только', 'ее', 'мне', 'было', 'вот', 'от', 'меня', 'еще', 'нет', 'о', 'из',
        'ему', 'теперь', 'когда', 'даже', 'ну', 'вдруг', 'ли', 'если', 'уже', 'или',
        'ни', 'быть', 'был', 'него', 'до', 'вас', 'нибудь', 'опять', 'уж', 'вам', 'ведь',
        'там', 'потом', 'себя', 'ничего', 'ей', 'может', 'они', 'тут', 'где', 'есть',
        'надо', 'ней', 'для', 'мы', 'тебя', 'их', 'чем', 'была', 'сам', 'чтоб', 'без',
        'будто', 'чего', 'раз', 'тоже', 'себе', 'под', 'будет', 'ж', 'тогда', 'кто',
        'этот', 'того', 'потому', 'этого', 'какой', 'совсем', 'ним', 'здесь', 'этом'
    }

def tokenize(self, text: str) -> List[str]:
    """Токенизация текста"""
    if not text:
        return []

    # Приводим к нижнему регистру
    text = text.lower()

    # Удаляем специальные символы
    text = re.sub(r'[^\w\s]', ' ', text)

    # Разбиваем на слова
    words = text.split()

    # Удаляем стоп-слова
    words = [word for word in words if word not in self._stop_words]
```

```
        return words

    def extract_keywords(self, text: str, max_keywords: int = 10) -> List[str]:
        """Извлечение ключевых слов из текста"""
        words = self.tokenize(text)

        if not words:
            return []

        # Подсчет частот
        from collections import Counter
        word_freq = Counter(words)

        # Берем самые частые слова
        common_words = word_freq.most_common(max_keywords)

        return [word for word, freq in common_words]
```

2. Интеграция с PHP-проектом

examples/php_integration.php

php

```
<?php
/**
 * Пример интеграции Python-модуля с PHP-проектом
 */

class PythonSimilarityAnalyzer
{
    private $pythonScriptPath;
    private $pythonExecutable;

    public function __construct(string $pythonScriptPath = null)
    {
```

```
        $this->pythonScriptPath = $pythonScriptPath ?? __DIR__ . '/../book-similarity';
        $this->pythonExecutable = 'python3';
    }

    /**
     * Вызов Python-анализатора через командную строку
     */
    public function analyzeSimilarity(array $book1, array $book2): array
    {
        $data = [
            'book1' => $book1,
            'book2' => $book2,
            'action' => 'analyze'
        ];

        $jsonData = json_encode($data, JSON_UNESCAPED_UNICODE);
        $jsonDataEscaped = escapeshellarg($jsonData);

        $command = sprintf(
            '%s -c "import sys; sys.path.append(\'%s\'); from examples.standalone_analyzer
import main; main()" --json %s',
            $this->pythonExecutable,
            $this->pythonScriptPath,
            $jsonDataEscaped
        );

        $output = shell_exec($command);

        if (!$output) {
            return ['error' => 'Python script execution failed'];
        }
```

```
        return json_decode($output, true) ?? ['error' => 'Invalid JSON response'];
    }

    /**
     * Поиск похожих книг
     */
    public function findSimilarBooks(int $bookId, array $bookList, float $threshold = 0.7): array
    {
        $data = [
            'action' => 'find_similar',
            'book_id' => $bookId,
            'books' => $bookList,
            'threshold' => $threshold
        ];

        $jsonData = escapeshellarg(json_encode($data, JSON_UNESCAPED_UNICODE));

        $command = sprintf(
            '%s -c "import sys; sys.path.append(\'%s\'); from examples.standalone_analyzer import main; main()" --json %s',
            $this->pythonExecutable,
            $this->pythonScriptPath,
            $jsonData
        );

        $output = shell_exec($command);

        return $output ? json_decode($output, true) : [];
    }

    /**
```

```
     * Генерация рекомендаций
     */
    public function getRecommendations(array $userBooks, array $allBooks, array $preferences = []): array
    {
        $data = [
            'action' => 'recommend',
            'user_books' => $userBooks,
            'all_books' => $allBooks,
            'preferences' => $preferences
        ];

        $jsonData = escapeshellarg(json_encode($data, JSON_UNESCAPED_UNICODE));

        $command = sprintf(
            '%s %s/examples/standalone_analyzer.py --json %s',
            $this->pythonExecutable,
            $this->pythonScriptPath,
            $jsonData
        );

        $output = shell_exec($command);

        return $output ? json_decode($output, true) : [];
    }

    /**
     * Анализ метрик кода PHP-классов
     */
    public function analyzeCodeMetrics(string $phpCode): array
    {
        $data = [
```

```
            'action' => 'analyze_metrics',
            'php_code' => $phpCode
        ];

        $tempFile = tempnam(sys_get_temp_dir(), 'php_code_');
        file_put_contents($tempFile, json_encode($data));

        $command = sprintf(
            '%s %s/examples/code_analyzer.py --input %s',
            $this->pythonExecutable,
            $this->pythonScriptPath,
            escapeshellarg($tempFile)
        );

        $output = shell_exec($command);

        unlink($tempFile);

        return $output ? json_decode($output, true) : [];
    }
}

// Пример использования в PHP-проекте
$analyzer = new PythonSimilarityAnalyzer();

// Анализ схожести двух книг
$book1 = [
    'id' => 1,
    'title' => 'Программирование на PHP',
    'author' => 'Иванов И.И.',
    'description' => 'Книга о программировании на PHP',
    'keywords' => ['php', 'программирование', 'web']
```

```
];

$book2 = [
    'id' => 2,
    'title' => 'PHP для начинающих',
    'author' => 'Иванов Иван',
    'description' => 'Введение в программирование на PHP',
    'keywords' => ['php', 'начало', 'программирование']
];

$similarity = $analyzer->analyzeSimilarity($book1, $book2);

echo "Схожесть книг: " . json_encode($similarity, JSON_PRETTY_PRINT | JSON_UNESCAPED_UNICODE);
```

examples/standalone_analyzer.py

```
python
#!/usr/bin/env python3
"""
Скрипт для запуска из PHP через командную строку
"""
import json
import sys
import os

# Добавляем путь к модулю
sys.path.append(os.path.dirname(os.path.dirname(os.path.abspath(__file__))))

from similarity.analyzer import BookSimilarityAnalyzer, BookData


def main():
    """Основная функция для запуска из командной строки"""
```

```
import argparse

parser = argparse.ArgumentParser(description='Анализатор схожести книг')
parser.add_argument('--json', type=str, help='JSON данные для анализа')
parser.add_argument('--action', type=str, default='analyze',
                    choices=['analyze', 'find_similar', 'recommend'])

args = parser.parse_args()

if args.json:
    data = json.loads(args.json)
    action = data.get('action', 'analyze')
else:
    # Чтение из stdin
    input_data = sys.stdin.read()
    if input_data:
        data = json.loads(input_data)
        action = data.get('action', 'analyze')
    else:
        print(json.dumps({'error': 'No input data provided'}))
        return

analyzer = BookSimilarityAnalyzer()

if action == 'analyze':
    # Анализ схожести двух книг
    book1_data = data.get('book1', {})
    book2_data = data.get('book2', {})

    book1 = BookData(
        id=book1_data.get('id', 0),
        title=book1_data.get('title', ''),
```

```
            author=book1_data.get('author', ''),
            description=book1_data.get('description', ''),
            keywords=book1_data.get('keywords', []),
            category=book1_data.get('category', ''),
            year=book1_data.get('year', 0),
            isbn=book1_data.get('isbn', '')
        )

        book2 = BookData(
            id=book2_data.get('id', 0),
            title=book2_data.get('title', ''),
            author=book2_data.get('author', ''),
            description=book2_data.get('description', ''),
            keywords=book2_data.get('keywords', []),
            category=book2_data.get('category', ''),
            year=book2_data.get('year', 0),
            isbn=book2_data.get('isbn', '')
        )

        similarity = analyzer.calculate_similarity(book1, book2, "combined")
        print(json.dumps(similarity, ensure_ascii=False))

    elif action == 'find_similar':
        # Поиск похожих книг
        target_id = data.get('book_id')
        book_list_data = data.get('books', [])
        threshold = data.get('threshold', 0.7)

        # Находим целевую книгу
        target_book = None
        all_books = []
```

```
for book_data in book_list_data:
    book = BookData(
        id=book_data.get('id', 0),
        title=book_data.get('title', ''),
        author=book_data.get('author', ''),
        description=book_data.get('description', ''),
        keywords=book_data.get('keywords', []),
        category=book_data.get('category', ''),
        year=book_data.get('year', 0),
        isbn=book_data.get('isbn', '')
    )

    if book.id == target_id:
        target_book = book

    all_books.append(book)

if not target_book:
    print(json.dumps({'error': 'Target book not found'}))
    return

similar_books = analyzer.find_similar_books(target_book, all_books, threshold)

result = []
for similar_book, similarity_data in similar_books:
    result.append({
        'book': {
            'id': similar_book.id,
            'title': similar_book.title,
            'author': similar_book.author
        },
        'similarity': similarity_data
```

```
        })

    print(json.dumps(result, ensure_ascii=False))

elif action == 'recommend':
    # Генерация рекомендаций
    user_books_data = data.get('user_books', [])
    all_books_data = data.get('all_books', [])
    preferences = data.get('preferences', {})

    user_books = []
    for book_data in user_books_data:
        user_books.append(BookData(
            id=book_data.get('id', 0),
            title=book_data.get('title', ''),
            author=book_data.get('author', ''),
            description=book_data.get('description', ''),
            keywords=book_data.get('keywords', []),
            category=book_data.get('category', ''),
            year=book_data.get('year', 0),
            isbn=book_data.get('isbn', '')
        ))

    all_books = []
    for book_data in all_books_data:
        all_books.append(BookData(
            id=book_data.get('id', 0),
            title=book_data.get('title', ''),
            author=book_data.get('author', ''),
            description=book_data.get('description', ''),
            keywords=book_data.get('keywords', []),
            category=book_data.get('category', ''),
```

```
                year=book_data.get('year', 0),
                isbn=book_data.get('isbn', '')
            ))

        recommendations = analyzer.get_recommendations(user_books, all_books, preferences)

        result = []
        for book in recommendations:
            result.append({
                'id': book.id,
                'title': book.title,
                'author': book.author,
                'category': book.category,
                'year': book.year
            })

        print(json.dumps(result, ensure_ascii=False))

    else:
        print(json.dumps({'error': f'Unknown action: {action}'}))


if __name__ == '__main__':
    main()
```

3. Тесты для модуля

tests/test_analyzer.py

```
python
import unittest
import sys
import os
```

```
sys.path.append(os.path.dirname(os.path.dirname(os.path.abspath(__file__))))

from similarity.analyzer import BookSimilarityAnalyzer, BookData


class TestBookSimilarityAnalyzer(unittest.TestCase):

    def setUp(self):
        self.analyzer = BookSimilarityAnalyzer()

        self.book1 = BookData(
            id=1,
            title="Программирование на Python",
            author="Иванов И.И.",
            description="Книга о программировании на языке Python",
            keywords=["python", "программирование", "код"],
            category="Программирование",
            year=2020,
            isbn="978-5-4461-1234-5"
        )

        self.book2 = BookData(
            id=2,
            title="Python для начинающих",
            author="Иванов Иван Иванович",
            description="Введение в программирование на Python",
            keywords=["python", "начало", "обучение"],
            category="Программирование",
            year=2021,
            isbn="978-5-4461-5678-9"
        )
```

```
        self.book3 = BookData(
            id=3,
            title="История России",
            author="Петров П.П.",
            description="Полная история России",
            keywords=["история", "россия", "страна"],
            category="История",
            year=2015,
            isbn="978-5-271-2345-6"
        )

    def test_title_similarity_identical(self):
        similarity = self.analyzer.calculate_similarity(self.book1, self.book1, "title")
        self.assertIn('title_similarity', similarity)
        self.assertEqual(similarity['title_similarity'], 1.0)

    def test_title_similarity_similar(self):
        similarity = self.analyzer.calculate_similarity(self.book1, self.book2, "title")
        self.assertIn('title_similarity', similarity)
        self.assertGreater(similarity['title_similarity'], 0.5)

    def test_title_similarity_different(self):
        similarity = self.analyzer.calculate_similarity(self.book1, self.book3, "title")
        self.assertIn('title_similarity', similarity)
        self.assertLess(similarity['title_similarity'], 0.3)

    def test_author_similarity_identical(self):
        similarity = self.analyzer.calculate_similarity(self.book1, self.book1, "author")
        self.assertIn('author_similarity', similarity)
        self.assertEqual(similarity['author_similarity'], 1.0)

    def test_author_similarity_similar(self):
```

```
        similarity = self.analyzer.calculate_similarity(self.book1, self.book2, "author")
        self.assertIn('author_similarity', similarity)
        self.assertGreater(similarity['author_similarity'], 0.7)

    def test_content_similarity(self):
        similarity = self.analyzer.calculate_similarity(self.book1, self.book2, "content")
        self.assertIn('content_similarity', similarity)
        self.assertGreater(similarity['content_similarity'], 0.0)

    def test_jaccard_similarity(self):
        similarity = self.analyzer.calculate_similarity(self.book1, self.book2, "jaccard")
        self.assertIn('jaccard_similarity', similarity)
        self.assertGreater(similarity['jaccard_similarity'], 0.0)

    def test_combined_similarity(self):
        similarity = self.analyzer.calculate_similarity(self.book1, self.book2, "combined")
        self.assertIn('combined_similarity', similarity)
        self.assertIn('similarity_level', similarity)
        self.assertGreater(similarity['combined_similarity'], 0.0)

    def test_find_similar_books(self):
        books = [self.book1, self.book2, self.book3]
        similar_books = self.analyzer.find_similar_books(self.book1, books, threshold=0.5)

        self.assertGreater(len(similar_books), 0)

        # Книга 2 должна быть похожа на книгу 1
        found_book2 = False
        for book, similarity in similar_books:
            if book.id == self.book2.id:
                found_book2 = True
                self.assertGreater(similarity.get('combined_similarity', 0), 0.5)
```

```
                break

        self.assertTrue(found_book2, "Book 2 should be found as similar to Book 1")

    def test_get_recommendations(self):
        user_books = [self.book1]
        all_books = [self.book1, self.book2, self.book3]

        recommendations = self.analyzer.get_recommendations(user_books, all_books)

        # Должна быть рекомендована книга 2, но не книга 1 (уже есть у пользователя) и не книга 3 (не похожа)
        self.assertEqual(len(recommendations), 1)
        self.assertEqual(recommendations[0].id, self.book2.id)

    def test_cache_functionality(self):
        # Первый вызов
        similarity1 = self.analyzer.calculate_similarity(self.book1, self.book2, "combined")

        # Второй вызов должен использовать кэш
        similarity2 = self.analyzer.calculate_similarity(self.book1, self.book2, "combined")

        stats = self.analyzer.get_statistics()
        self.assertGreater(stats['cache_hits'], 0)

    def test_empty_input(self):
        empty_book = BookData(id=0, title="", author="")
        similarity = self.analyzer.calculate_similarity(empty_book, self.book1, "combined")
        self.assertIn('combined_similarity', similarity)
        self.assertEqual(similarity['combined_similarity'], 0.0)
```

```
if __name__ == '__main__':
    unittest.main()
```

tests/test_integration.py

```
python
import unittest
import json
import tempfile
import os
import sys

sys.path.append(os.path.dirname(os.path.dirname(os.path.abspath(__file__))))

from examples.standalone_analyzer import main


class TestIntegration(unittest.TestCase):

    def test_standalone_analyzer_stdin(self):
        """Тестирование работы через stdin"""
        test_data = {
            'action': 'analyze',
            'book1': {
                'id': 1,
                'title': 'Test Book 1',
                'author': 'Author One',
                'description': 'Test description one'
            },
            'book2': {
                'id': 2,
                'title': 'Test Book 2',
                'author': 'Author Two',
                'description': 'Test description two'
```

```
    }
}

# Создаем временный файл с тестовыми данными
with tempfile.NamedTemporaryFile(mode='w', suffix='.json', delete=False) as f:
    json.dump(test_data, f)
    temp_file = f.name

try:
    # Читаем данные из файла как бы из stdin
    import io
    from contextlib import redirect_stdout

    with open(temp_file, 'r') as f:
        input_data = f.read()

    # Захватываем вывод
    output = io.StringIO()
    with redirect_stdout(output):
        # Имитируем вызов main с данными
        import argparse

        class Args:
            json = None
            action = 'analyze'

        args = Args()

        # В реальном тесте нужно мокировать sys.stdin
        import sys
        original_stdin = sys.stdin
        sys.stdin = io.StringIO(input_data)
```

```
            try:
                main()
            finally:
                sys.stdin = original_stdin

        result = output.getvalue()

        # Проверяем, что результат - валидный JSON
        parsed_result = json.loads(result)
        self.assertIn('combined_similarity', parsed_result)

    finally:
        os.unlink(temp_file)


if __name__ == '__main__':
    unittest.main()
```

4. Файлы конфигурации

setup.py

```
python
from setuptools import setup, find_packages

setup(
    name="book-similarity-analyzer",
    version="1.0.0",
    author="Smart Library Team",
    description="Python module for book similarity analysis (for integration with PHP)",
    packages=find_packages(),
    install_requires=[
        # Минимальные зависимости
    ],
```

```
    extras_require={
        'dev': [
            'pytest>=7.0.0',
            'pytest-cov>=4.0.0',
            'flake8>=6.0.0',
            'pylint>=3.0.0',
            'mypy>=1.0.0',
        ],
        'analysis': [
            'scipy>=1.10.0',  # Для статистического анализа
            'numpy>=1.24.0',  # Для численных вычислений
        ]
    },
    python_requires=">=3.8",
    classifiers=[
        "Development Status :: 4 - Beta",
        "Intended Audience :: Developers",
        "Topic :: Software Development :: Libraries :: Python Modules",
        "Programming Language :: Python :: 3",
        "Programming Language :: Python :: 3.8",
        "Programming Language :: Python :: 3.9",
        "Programming Language :: Python :: 3.10",
        "Programming Language :: Python :: 3.11",
    ],
)
```

requirements.txt

```
# Основные зависимости
# (можно оставить пустым для минимальной установки)
# Дополнительные зависимости для анализа
# scipy>=1.10.0
# numpy>=1.24.0
```

Дополнительные зависимости для тестирования

pytest>=7.0.0

pytest-cov>=4.0.0

flake8>=6.0.0

README.md

markdown

Book Similarity Analyzer (Python Module)

Модуль на Python для анализа схожести книг, предназначенный для интеграции с PHP-проектом "Умная библиотека".

Особенности

- Анализ схожести книг по названию, автору, содержанию и ключевым словам
- Несколько алгоритмов сравнения (Жаккард, косинусная схожесть, расстояние Левенштейна)
- Генерация рекомендаций на основе схожести
- Кэширование результатов для производительности
- Специально содержит "проблемный" код для анализа метриками

Интеграция с PHP

Модуль может быть вызван из PHP через командную строку:

php

$analyzer = new PythonSimilarityAnalyzer();

$similarity = $analyzer->analyzeSimilarity($book1, $book2);

Установка

1. Клонировать модуль в директорию PHP-проекта
2. Установить Python 3.8 или выше
3. (Опционально) Установить зависимости: pip install -r requirements.txt

Структура модуля

book-similarity/

├── similarity/ # Основной код модуля

├── tests/ # Тесты

├── examples/ # Примеры использования

└── setup.py # Конфигурация пакета

Использование в PHP

php

```php
// Создание анализатора
$analyzer = new PythonSimilarityAnalyzer('/path/to/book-similarity');

// Анализ схожести двух книг
$similarity = $analyzer->analyzeSimilarity($book1, $book2);

// Поиск похожих книг
$similarBooks = $analyzer->findSimilarBooks(123, $bookList, 0.7);

// Генерация рекомендаций
$recommendations = $analyzer->getRecommendations($userBooks, $allBooks, $preferences);
```

Тестирование

bash

```bash
# Запуск всех тестов
python -m pytest tests/

# Запуск тестов с покрытием
python -m pytest --cov=similarity tests/

# Проверка стиля кода
flake8 similarity/
pylint similarity/
```

Особенности для анализа метрик

Модуль специально содержит:

- Высокую цикломатическую сложность методов
- Дублирование кода
- Сложные вложенные условия
- Длинные методы
- Сильную связанность

Это позволяет использовать его для обучения анализу метрик кода.

Этот Python-модуль можно легко интегрировать в существующий PHP-проект через вызов командной строки. Модуль выполняет полезную функцию (анализ схожести книг) и при этом содержит намеренные проблемы для анализа метриками, что делает его идеальным для учебных целей.

Составитель
Витвицкий Максим Николаевич

Методические указания по выполнению самостоятельной работы
для студентов очной формы обучения
по направлению специальности
09.02.07 «Информационные системы и программирование»

Публикуется в авторской редакции