МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Институт информационных технологий (ИИТ)
Кафедра математического обеспечения и стандартизации информационных технологий (МОСИТ)

ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ №3

по дисциплине «Тестирование и верификация программного обеспечения»

Студент группы *ИКБО-50-23. Павлов Н.С..* _____
(подпись)

Преподаватель *Ильичев Г.П.* _____
(подпись)

Москва 2025 г.

СОДЕРЖАНИЕ

1. ПОСТАНОВКА ЗАДАЧИ

Цель работы: изучение и применение различных подходов к разработке программного обеспечения, основанного на тестировании, для повышения качества, надёжности и поддерживаемости кода.

Задачи:

1. Изучить теоретические основы методологий тестирования: TDD, ATDD, BDD и SDD;
2. Исследовать преимущества и недостатки каждого подхода;
3. Реализовать практические примеры для каждого метода;
4. Проанализировать влияние интеграции тестирования на архитектуру и качество программного продукта;
5. Подготовить итоговый отчёт с выводами и рекомендациями по интеграции подходов в современные процессы разработки.

Персональный вариант 47: Органайзер музыкальной библиотеки

Функции — добавление трека, редактирование метаданных, поиск по песням.

- TDD — тесты для функций добавления, поиска треков и изменения метаданных.
- ATDD — приёмочные сценарии для управления музыкальной коллекцией.
- BDD — сценарий «Given трек добавлен, When ищу по названию, Then трек найден».
- SDD — таблицы с метаданными треков и ожидаемыми результатами поиска.

2 ВЫПОЛНЕНИЕ РАБОТЫ

2.1 РЕАЛИЗАЦИЯ С ПОМОЩЬЮ TDD (TEST-DRIVEN DEVELOPMENT)

На основе описанных требований задания были созданы тесты с использованием библиотеки Python unitest (Листинг 1), которые описывают поведение основного функционала разрабатываемой программы, а именно функций добавления и поиска треков в плейлисте, а также изменения их метаданных.

Листинг 1 – Unitest функций добавления, поиска треков и изменения метаданных(TDD.py)

```
import unittest
from music_library import MusicLibrary, Track


class TestMusicLibrary(unittest.TestCase):
    def setUp(self):
        """Подготовка объектов для тестов"""
        self.library = MusicLibrary()
        self.track_1 = Track(
            title="Конь",
            artist="Любэ",
            genre="Фолк-рок",
            year=1994,
            duration=218
        )
        self.track_2 = Track(
            title="А зори здесь тихие",
            artist="Любэ",
            genre="Русская эстрада",
            year=2015,
            duration=214
        )
        self.track_3 = Track(
            title="Кукла колдуна",
            artist="Король и Шут",
            genre="Фолк-рок",
            year=1998,
            duration=203
        )
        self.track_4 = Track(
            title="Кукла",
            artist="Иванушки International",
            genre="Поп",
            year=1997,
            duration=314
        )

    def test_add_track(self):
        """Тест на добавление трека в библиотеку"""
        self.library.add_track(self.track_1)
        self.assertIn(self.track_1, self.library.tracks)
        self.assertEqual(len(self.library.tracks), 1)
```

```
    def test_add_multiple_tracks(self):
        """Тест на добавление нескольких треков"""
        self.library.add_track(self.track_1)
        self.library.add_track(self.track_2)
        self.assertEqual(len(self.library.tracks), 2)
        self.assertIn(self.track_1, self.library.tracks)
        self.assertIn(self.track_2, self.library.tracks)

    def test_add_copy_track(self):
        """Тест на добавление уже существующего трека"""
        self.library.add_track(self.track_1)
        self.library.add_track(self.track_1)
        self.assertEqual(len(self.library.tracks), 1)

    def test_find_track_by_title_exact_match(self):
        """Тест на поиск трека по точному названию"""
        self.library.add_track(self.track_1)
        self.library.add_track(self.track_2)

        found_tracks = self.library.find_by_title("Конь")
        self.assertIn(self.track_1, found_tracks)
        self.assertEqual(len(found_tracks), 1)

    def test_find_track_by_title_partial_match(self):
        """Тест на поиск по части названия"""
        self.library.add_track(self.track_2)
        self.library.add_track(self.track_3)
        self.library.add_track(self.track_4)

        found_tracks = self.library.find_by_title("Кукла")
        self.assertIn(self.track_3, found_tracks)
        self.assertIn(self.track_4, found_tracks)
        self.assertEqual(len(found_tracks), 2)

    def test_find_track_by_title_case_insensitive(self):
        """Тест на поиск без учёта регистра"""
        self.library.add_track(self.track_2)

        found_tracks = self.library.find_by_title("а зори здесь тихие")
        self.assertIn(self.track_2, found_tracks)
        self.assertEqual(len(found_tracks), 1)

    def test_find_track_by_title_not_found(self):
        """Тест на поиск несуществующего трека"""
        self.library.add_track(self.track_1)

        found_tracks = self.library.find_by_title("Несуществующая песня")
        self.assertEqual(len(found_tracks), 0)

    def test_find_by_artist(self):
        """Тест поиска по артисту"""
        self.library.add_track(self.track_1)
        self.library.add_track(self.track_2)
        self.library.add_track(self.track_3)

        found_tracks = self.library.find_by_artist("Любэ")
        self.assertIn(self.track_1, found_tracks)
        self.assertIn(self.track_2, found_tracks)
        self.assertEqual(len(found_tracks), 2)
```

```
    def test_find_by_genre(self):
        """Тест поиска по жанру"""
        self.library.add_track(self.track_1)
        self.library.add_track(self.track_3)
        self.library.add_track(self.track_4)

        found_tracks = self.library.find_by_genre("Рок")
        self.assertIn(self.track_1, found_tracks)
        self.assertIn(self.track_3, found_tracks)
        self.assertEqual(len(found_tracks), 2)

    def test_edit_track_metadata_artist(self):
        """Тест на редактирование артиста"""
        self.library.add_track(self.track_2)

        new_artist = "Любэ, Алексей Филатов"
        self.library.edit_track_metadata(self.track_2.title,
artist=new_artist)

        updated_track = self.library.find_by_title("А зори здесь тихие")[0]
        self.assertEqual(updated_track.artist, new_artist)

    def test_edit_track_metadata_genre(self):
        """Тест на редактирование жанра"""
        self.library.add_track(self.track_4)

        new_genre = "Поп-музыка"
        self.library.edit_track_metadata(self.track_4.title, genre=new_genre)

        updated_track = self.library.find_by_title("Кукла")[0]
        self.assertEqual(updated_track.genre, new_genre)

    def test_edit_track_metadata_multiple_fields(self):
        """Тест на редактирование нескольких полей одновременно"""
        self.library.add_track(self.track_1)

        self.library.edit_track_metadata(
            self.track_1.title,
            genre="Эстрада, фолк-рок",
            year=1995,
            duration=246
        )
        updated_track = self.library.find_by_title("Конь")[0]
        self.assertEqual(updated_track.genre, "Эстрада, фолк-рок")
        self.assertEqual(updated_track.year, 1995)
        self.assertEqual(updated_track.duration, 246)


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

Запустим наши тесты, чтобы убедиться, что они провалятся, так как сам функционал еще не реализован.

Результат: *ModuleNotFoundError: No module named 'music_library'* — это ожидаемо, мы ещё не создали модуль.

Теперь напишем минимальный код, чтобы тесты прошли. Создадим классы *Track* и *MusicLibrary* с необходимыми методами (Листинг 2).

Листинг 2 – Код, покрывающий все unitest (music_library.py)

```
class Track:
    def __init__(self, title, artist, genre=None, year=None, duration=None):
        self.title = title
        self.artist = artist
        self.genre = genre
        self.year = year
        self.duration = duration


class MusicLibrary:
    def __init__(self):
        self.tracks = []

    def add_track(self, track):
    """Добавление трека в библиотеку с проверкой на дубликаты"""
    for existing_track in self.tracks:
        if (existing_track.title.lower() == track.title.lower() and
            existing_track.artist.lower() == track.artist.lower() and
            existing_track.genre == track.genre and
            existing_track.year == track.year and
            existing_track.duration == track.duration):
            return
    self.tracks.append(track)



    def find_by_title(self, title):
        """Поиск треков по названию"""
        title_lower = title.lower()
        return [track for track in self.tracks if title_lower in
track.title.lower()]

    def find_by_artist(self, artist):
        """Поиск треков по артисту"""
        artist_lower = artist.lower()
        return [track for track in self.tracks if artist_lower in
track.artist.lower()]

    def find_by_genre(self, genre):
        """Поиск треков по жанру"""
        genre_lower = genre.lower()
        return [track for track in self.tracks if track.genre and genre_lower
in track.genre.lower()]

    def edit_track_metadata(self, title, **kwargs):
        """Редактирование метаданных трека"""
        tracks_to_edit = self.find_by_title(title)

        for track in tracks_to_edit:
            for key, value in kwargs.items():
                if hasattr(track, key) and value is not None:
                    setattr(track, key, value)
```

Повторно запустим ранее созданные тесты и убедимся в правильности и полноте написанного кода.

Все тесты оказались успешно пройдены (рис. 1), поэтому рефакторинг кода не требуется.

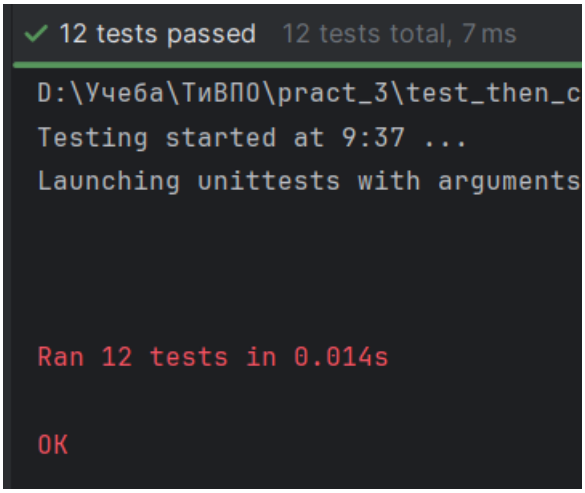

Рисунок 1 – Результаты тестирования

2.2 РЕАЛИЗАЦИЯ С ПОМОЩЬЮ ATDD (ACCEPTANCE TEST-DRIVEN DEVELOPMENT)

Проанализировав требования поставленной задачи, были составлены сценарии использования приложения:

Сценарий 1. Добавление трека в библиотеку

Предусловие: пользователь заходит в раздел «Добавить трек».

Действия:

1. пользователь вводит название трека: «Конь»;
2. указывает артиста: «Любэ»;
3. выбирает жанр: «Фолк-рок»;
4. устанавливает год выпуска: 1994;
5. указывает длительность: 218 секунд;
6. нажимает кнопку «Сохранить».

Ожидаемый результат: трек «Конь» появляется в списке треков библиотеки.

Сценарий 2. Поиск треков по артисту

Предусловие: в библиотеке есть треки разных артистов.

Действия:

1. пользователь вводит в поиск имя артиста: «Любэ»;
2. нажимает кнопку «Найти».

Ожидаемый результат: система показывает все треки артиста «Любэ» («Конь», «А зори здесь тихие»).

Сценарий 3. Фильтрация треков по жанру

Предусловие: в библиотеке есть треки разных жанров.

Действия:

1. пользователь выбирает фильтр «Жанр»;
2. выбирает значение «Фолк-рок»;
3. применяет фильтр.

Ожидаемый результат: система показывает все треки жанра «Фолк-рок» («Конь» - Любэ, «Кукла колдуна» - Король и Шут).

Сценарий 4. Поиск треков по части названия

Предусловие: в библиотеке есть треки с похожими названиями.

Действия:

1. пользователь вводит в поиск: «Кукла»;
2. нажимает кнопку «Найти».

Ожидаемый результат: система показывает все треки, содержащие «Кукла» в названии («Кукла колдуна», «Кукла»).

Сценарий 5. Редактирование метаданных трека

Предусловие: в библиотеке есть трек «Конь» с неполной информацией.

Действия:

1. пользователь выбирает трек «Конь»;
2. нажимает кнопку «Редактировать»;
3. изменяет жанр на «Фолк-рок, военная песня»;
4. устанавливает год: 1994;
5. изменяет длительность: 220 секунд;
6. нажимает кнопку «Сохранить».

Ожидаемый результат: информация о треке «Конь» обновлена с новыми метаданными.

Сценарий 6. Поиск без учёта регистра

Предусловие: в библиотеке есть трек «А зори здесь тихие».

Действия:

1. пользователь вводит в поиск: «а зори здесь тихие» (в нижнем регистре);
2. нажимает кнопку «Найти».

Ожидаемый результат: система находит трек «А зори здесь тихие» независимо от регистра.

Сценарий 7. Предотвращение добавления дубликатов

Предусловие: в библиотеке уже есть трек «Конь» (Любэ, Фолк-рок, 1994, 218).

Действия:

1. пользователь пытается добавить трек с такими же данными: «Конь», «Любэ», «Фолк-рок», 1994, 218;

2. нажимает кнопку «Сохранить».

Ожидаемый результат: система отклоняет добавление дубликата, количество треков в библиотеке не изменяется.

Обозначенные сценарии были реализованы в виде unitest (Листинг 3) и протестированы на программе из Листинга 2. В результате все тесты завершились с успехом (рис. 2), рефакторинг кода не требуется.

Листинг 3 – Unitest для сценариев использования приложения (ATDD.py)

```
import unittest
from music_library import MusicLibrary, Track


class TestMusicLibraryAcceptance(unittest.TestCase):
    def setUp(self):
        self.library = MusicLibrary()

        self.tracks = [
            Track("Конь", "Любэ", "Фолк-рок", 1994, 218),
            Track("А зори здесь тихие", "Любэ", "Русская эстрада", 2015,
214),
            Track("Кукла колдуна", "Король и Шут", "Фолк-рок", 1998, 203),
            Track("Кукла", "Иванушки International", "Поп", 1997, 314)
        ]

    def test_acceptance_add_track_to_library(self):
        """Сценарий: Пользователь добавляет трек в библиотеку"""
        # Given - пользователь хочет добавить песню "Конь" группы Любэ
        new_track = Track("Конь", "Любэ", "Фолк-рок", 1994, 218)

        # When - пользователь добавляет трек в библиотеку
        self.library.add_track(new_track)

        # Then - трек успешно добавлен и доступен для поиска
        found_tracks = self.library.find_by_title("Конь")
        self.assertIn(new_track, found_tracks)
        self.assertEqual(len(found_tracks), 1)

    def test_acceptance_find_all_songs_by_artist(self):
        """Сценарий: Пользователь ищет все песни группы Любэ"""
        # Given - в библиотеке есть несколько песен разных артистов
        for track in self.tracks:
            self.library.add_track(track)

        # When - пользователь ищет все треки группы "Любэ"
        lyube_tracks = self.library.find_by_artist("Любэ")
```

```
        # Then - система возвращает все песни этого артиста
        self.assertEqual(len(lyube_tracks), 2)
        titles = set(track.title for track in lyube_tracks)
        expected_titles = {"Конь", "А зори здесь тихие"}
        self.assertEqual(titles, expected_titles)

    def test_acceptance_filter_songs_by_genre(self):
        """Сценарий: Пользователь фильтрует песни по жанру Фолк-рок"""
        # Given - библиотека содержит песни разных жанров
        for track in self.tracks:
            self.library.add_track(track)

        # When - пользователь выбирает жанр "Фолк-рок"
        folk_rock_tracks = self.library.find_by_genre("Фолк-рок")

        # Then - система показывает все треки этого жанра
        self.assertEqual(len(folk_rock_tracks), 2)
        artists = set(track.artist for track in folk_rock_tracks)
        self.assertEqual(artists, {"Любэ", "Король и Шут"})

    def test_acceptance_search_songs_with_similar_titles(self):
        """Сценарий: Пользователь ищет песни с похожими названиями"""
        # Given - в библиотеке есть песни с похожими названиями
        for track in self.tracks:
            self.library.add_track(track)

        # When - пользователь ищет по части названия "Кукла"
        doll_tracks = self.library.find_by_title("Кукла")

        # Then - система находит все подходящие треки
        self.assertEqual(len(doll_tracks), 2)
        titles = set(track.title for track in doll_tracks)
        expected_titles = {"Кукла колдуна", "Кукла"}
        self.assertEqual(titles, expected_titles)

    def test_acceptance_correct_track_metadata(self):
        """Сценарий: Администратор исправляет информацию о треке"""
        # Given - в библиотеке есть трек с неполной информацией
        track = Track("Конь", "Любэ", None, None, 218)
        self.library.add_track(track)

        # When - администратор обновляет метаданные трека
        self.library.edit_track_metadata(
            title="Конь",
            genre="Фолк-рок, военная песня",
            year=1994,
            duration=220
        )

        # Then - информация успешно обновляется
        updated_track = self.library.find_by_title("Конь")[0]
        self.assertEqual(updated_track.genre, "Фолк-рок, военная песня")
        self.assertEqual(updated_track.year, 1994)
        self.assertEqual(updated_track.duration, 220)

    def test_acceptance_case_insensitive_search(self):
        """Сценарий: Пользователь ищет песни без учёта регистра"""
        # Given - в библиотеке есть песни с разным регистром в названиях
        for track in self.tracks:
            self.library.add_track(track)

        # When - пользователь ищет "а зори здесь тихие" в нижнем регистре
        found_tracks = self.library.find_by_title("а зори здесь тихие")
```

```
        # Then - система находит песнь независимо от регистра
        self.assertEqual(len(found_tracks), 1)
        self.assertEqual(found_tracks[0].title, "А зори здесь тихие")

    def test_acceptance_prevent_duplicate_tracks(self):
        """Сценарий: Система предотвращает добавление дубликатов"""
        # Given - в библиотеке уже есть песня "Конь"
        original_track = Track("Конь", "Любэ", "Фолк-рок", 1994, 218)
        self.library.add_track(original_track)

        # When - пользователь пытается добавить тот же трек повторно
        duplicate_track = Track("Конь", "Любэ", "Фолк-рок", 1994, 218)
        self.library.add_track(duplicate_track)

        # Then - система отклоняет дубликат
        self.assertEqual(len(self.library.tracks), 1)


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

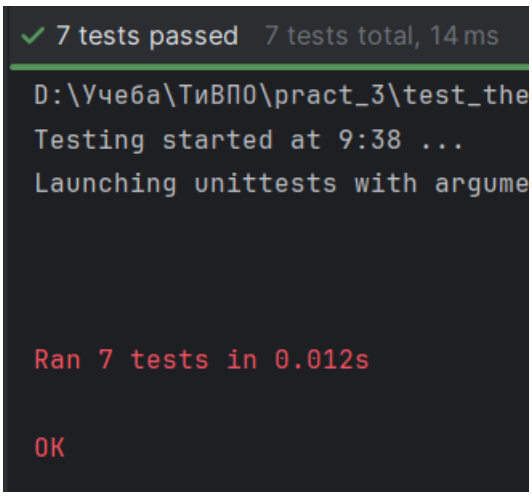

Рисунок 2 – Результаты тестирования

2.3 РЕАЛИЗАЦИЯ С ПОМОЩЬЮ BDD (BEHAVIOR-DRIVEN DEVELOPMENT)

Теперь перепишем сценарии из предыдущего шага на понятном для всех участников проекта языке. Для этого воспользуемся языком Gherkin, а именно его русифицированным вариантом. Полученные сценарии отображены в Листинге 4.

Листинг 4 – Сценарии использования приложения на языке Gherkin (BDD.feature)

```
# language: ru
Функция: Управление музыкальной библиотекой
  Чтобы организовать свою коллекцию треков
  Как пользователь музыкальной библиотеки
  Я хочу добавлять, искать и редактировать треки

  Сценарий: Поиск треков по названию
    Дано в библиотеке есть треки:
      | Название                | Артист                  | Жанр            |
      | Конь                    | Любэ                    | Фолк-рок        |
      | А зори здесь тихие      | Любэ                    | Русская эстрада|
      | Кукла колдуна           | Король и Шут            | Фолк-рок        |
    Когда я ищу треки по названию "Кукла"
    Тогда я должен найти 1 трек
    И среди найденных должны быть "Кукла колдуна"
```

Реализацию этих сценариев необходимо автоматизировать, поэтому воспользуемся инструментом для автоматизации тестов в Python – Behave. Шаги для автоматизации тестирования приведены в Листинге 5.

Запустим тесты для программы из Листинга 2 и убедимся в корректности их выполнения.

В результате все тесты успешно пройдены (рис. 3) и рефакторинг кода не требуется.

Листинг 5 – Шаги автоматизации тестирования Behave (BDD.py)

```
from behave import given, when, then
from music_library import MusicLibrary, Track


@given('в библиотеке есть треки:')
def step_impl(context):
    context.library = MusicLibrary()
    for row in context.table:
        title = row['Название']
        artist = row['Артист']
        genre = row.get('Жанр', None)
        year = int(row['год']) if row.get('год') else None
        duration = int(row['длительность']) if row.get('длительность') else
None
```

```
        track = Track(title, artist, genre, year, duration)
        context.library.add_track(track)


@when('я ищу треки по названию "{title}"')
def step_impl(context, title):
    context.found_tracks = context.library.find_by_title(title)


@then('я должен найти {count:d} трек')
def step_impl(context, count):
    assert len(context.found_tracks) == count, f"Ожидалось {count} треков, но
найдено {len(context.found_tracks)}"


@then('среди найденных должны быть "{title}"')
def step_impl(context, title):
    titles = [track.title for track in context.found_tracks]
    assert title in titles, f"Трек '{title}' не найден среди {titles}"
```

```
(.venv) PS D:\Учеба\ТиВПО\pract_3\test_then_code> behave
USING RUNNER: behave.runner:Runner
Функция: Управление музыкальной библиотекой # BDD.feature:2
  Чтобы организовать свою коллекцию треков
  Как пользователь музыкальной библиотеки
  Я хочу добавлять, искать и редактировать треки
  Сценарий: Поиск треков по названию              # BDD.feature:7
    Дано в библиотеке есть треки:                 # steps/BDD.py:5 0.000s
      | Название           | Артист       | Жанр            |
      | Конь               | Любэ         | Фолк-рок        |
      | А зори здесь тихие | Любэ         | Русская эстрада |
      | Кукла колдуна      | Король и Шут | Фолк-рок        |
    Когда я ищу треки по названию "Кукла"         # steps/BDD.py:19 0.000s
    Тогда я должен найти 1 трек                   # steps/BDD.py:24 0.000s
    И среди найденных должны быть "Кукла колдуна" # steps/BDD.py:29 0.000s

1 feature passed, 0 failed, 0 skipped
1 scenario passed, 0 failed, 0 skipped
4 steps passed, 0 failed, 0 skipped
Took 0min 0.000s
```

Рисунок 3 – Результаты тестирования

2.4 РЕАЛИЗАЦИЯ С ПОМОЩЬЮ SDD (SPECIFICATION BY EXAMPLE)

В проекте созданы спецификации требований таблицы с метаданными треков и ожидаемыми результатами поиска:

Таблица 1 – Метаданные добавленных треков

Название	Исполнитель	Жанр	Год	Время (в сек.)
Конь	Любэ	Фолк-рок	1994	218
А зори здесь тихие	Любэ	Русская эстрада	2015	214
Кукла колдуна	Король и Шут	Фолк-рок	1998	1998
Кукла	Иванушки International	Поп	1997	1997

Таблица 2 – Ожидаемые результаты поиска

Параметр поиска	Запрос	Ожидаемые результаты
По названию	Конь	Песня «Конь»
По названию	Кукла	Песни «Кукла колдуна» и «Кукла»
По названию	а зори здесь тихие	Песня «А зори здесь тихие»
По исполнителю	Король и Шут	Песня «Кукла колдуна»
По исполнителю	Любэ	Песни «Конь» и «»
По жанру	Поп	
По жанру	Фолк-рок	

Теперь напишем спецификацию по этим таблицам (Листинг 6) и реализуем автоматическое тестирование через Python unitest по созданной спецификации (Листинг 7).

Листинг 6 – Спецификация требований (SDD_specifications.py)

```
# Метаданные треков
TRACK_METADATA = [
    {
        "title": "Конь",
        "artist": "Любэ",
        "genre": "Фолк-рок",
        "year": 1994,
        "duration": 218
    },
    {
        "title": "А зори здесь тихие",
        "artist": "Любэ",
        "genre": "Русская эстрада",
        "year": 2015,
        "duration": 214
    },
    {
        "title": "Кукла колдуна",
        "artist": "Король и Шут",
        "genre": "Фолк-рок",
        "year": 1998,
        "duration": 203
    },
```

```
    {
        "title": "Кукла",
        "artist": "Иванушки International",
        "genre": "Поп",
        "year": 1997,
        "duration": 314
    }
]

# Ожидаемые результаты поиска
SEARCH_RESULTS = [
    {
        "search_type": "by_title",
        "query": "Конь",
        "expected_titles": ["Конь"]
    },
    {
        "search_type": "by_title",
        "query": "Кукла",
        "expected_titles": ["Кукла колдуна", "Кукла"]
    },
    {
        "search_type": "by_title",
        "query": "а зори здесь тихие",
        "expected_titles": ["А зори здесь тихие"]
    },
    {
        "search_type": "by_artist",
        "query": "Король и Шут",
        "expected_titles": ["Кукла колдуна"]
    },
    {
        "search_type": "by_artist",
        "query": "Любэ",
        "expected_titles": ["Конь", "А зори здесь тихие"]
    },
    {
        "search_type": "by_genre",
        "query": "Поп",
        "expected_titles": ["Кукла"]
    },
    {
        "search_type": "by_genre",
        "query": "Фолк-рок",
        "expected_titles": ["Конь", "Кукла колдуна"]
    }
]
```

Листинг 7 – Автоматизация тестирования спецификации через unitest (SDD.py)

```
import unittest
from music_library import MusicLibrary, Track
from SDD_specifications import TRACK_METADATA, SEARCH_RESULTS


class TestMusicLibrarySDD(unittest.TestCase):
    def setUp(self):
        """Настройка тестовой библиотеки на основе таблицы метаданных"""
        self.library = MusicLibrary()
```

```
        for track_data in TRACK_METADATA:
            track = Track(
                title=track_data['title'],
                artist=track_data['artist'],
                genre=track_data['genre'],
                year=track_data['year'],
                duration=track_data['duration']
            )
            self.library.add_track(track)

    def test_search_by_title_specifications(self):
        """Поиск по названию на основе таблицы спецификаций"""
        title_search_cases = [case for case in SEARCH_RESULTS if
case['search_type'] == 'by_title']

        for search_case in title_search_cases:
            with self.subTest(query=search_case['query']):
                results = self.library.find_by_title(search_case['query'])
                result_titles = [track.title for track in results]

                self.assertEqual(set(result_titles),
set(search_case['expected_titles']))

    def test_search_by_artist_specifications(self):
        """Поиск по артисту на основе таблицы спецификаций"""
        artist_search_cases = [case for case in SEARCH_RESULTS if
case['search_type'] == 'by_artist']

        for search_case in artist_search_cases:
            with self.subTest(query=search_case['query']):
                results = self.library.find_by_artist(search_case['query'])
                result_titles = [track.title for track in results]

                self.assertEqual(set(result_titles),
set(search_case['expected_titles']))

    def test_search_by_genre_specifications(self):
        """Поиск по жанру на основе таблицы спецификаций"""
        genre_search_cases = [case for case in SEARCH_RESULTS if
case['search_type'] == 'by_genre']

        for search_case in genre_search_cases:
            with self.subTest(query=search_case['query']):
                results = self.library.find_by_genre(search_case['query'])
                result_titles = [track.title for track in results]

                self.assertEqual(set(result_titles),
set(search_case['expected_titles']))


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

Запустим созданные тесты и убедимся в правильности и полноте написанного кода.

Все тесты оказались успешно пройдены (рис. 4), поэтому рефакторинг кода не требуется.

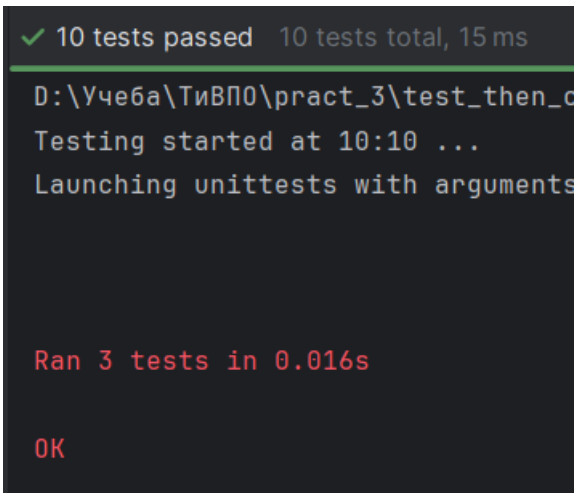

Рисунок 4 – Результаты тестирования

3 ЗАКЛЮЧЕНИЕ

В ходе выполнения практической работы был разработан программный продукт "Органайзер музыкальной библиотеки" и сопровождающая его документация. Общая оценка качества проведённой работы может быть сформулирована следующим образом.

1. Оценка качества программного продукта:

Программный модуль был успешно реализован с использованием методологии TDD, что позволило создать надежный и проверенный код. Все функциональные требования, включая добавление треков, поиск по различным критериям (название, исполнитель, жанр) и редактирование метаданных, были реализованы в полном объеме.

Качество кода подтверждено комплексным тестированием:

- Модульные тесты (TDD) обеспечили проверку корректности работы отдельных функций.
- Приёмочные тесты (ATDD) верифицировали соответствие системы пользовательским сценариям.
- Поведенческие тесты (BDD) подтвердили правильность бизнес-логики.
- Тестирование на основе спецификаций (SDD) гарантировало выполнение формальных требований.

Все тесты были пройдены успешно, что свидетельствует о высоком уровне надежности и соответствия программного продукта поставленным задачам.

2. Оценка качества документации:

Документация, представленная в отчете, является полной и структурированной. Она включает:

- Четкую постановку задачи с конкретизацией целей и функций
- Подробное описание процесса реализации для каждого подхода
- Листинги кода с комментариями
- Визуальные результаты тестирования
- Спецификации требований в табличной форме

3. Выводы о проделанной работе:

Проделанная работа позволила:

- Освоить практические навыки применения различных подходов тест-драйвенной разработки
- Сравнить эффективность TDD, ATDD, BDD и SDD методологий
- Создать качественный программный продукт с полным тестовым покрытием
- Сформировать комплексное понимание процессов обеспечения качества ПО

Наиболее ценным результатом стало демонстрация того, что интеграция различных подходов к тестированию на разных этапах разработки позволяет создать программный продукт, который не только технически корректно функционирует, но и полностью соответствует ожиданиям пользователей и формальным требованиям.