МИНОБРНАУКИ РОССИИ

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

Институт Информационных технологий

Кафедра Математического обеспечения и стандартизации информационных технологий

Отчет по практической работе №3

по дисциплине
«Тестирование и верификация программного обеспечения»
Тема: «Трекер здоровья»

Выполнил:
Студент группы ИКБО-62-23 Кокшарова А. А.

Принял: Петрова А. А.

МОСКВА 2025 г.

Оглавление

Введение

Современная разработка программного обеспечения все чаще опирается на методологии, основанные на тестировании, которые позволяют создавать более надежные и соответствующие требованиям системы. Такие подходы, как TDD (Test-Driven Development), ATDD (Acceptance Test-Driven Development), BDD (Behavior-Driven Development) и SDD (Specification by Example), помогают выстраивать процесс разработки таким образом, чтобы тестирование становилось неотъемлемой его частью.

В данной работе рассматривается практическое применение этих методологий на примере разработки приложения "Трекер здоровья". Последовательное использование различных подходов к тестированию позволяет комплексно оценить их влияние на качество кода, архитектуру приложения и процесс разработки в целом.

Цель работы

Изучение и применение различных подходов к разработке программного обеспечения, основанного на тестировании, для повышения качества, надёжности и поддерживаемости кода.

Для достижения поставленной цели работы студентам необходимо выполнить ряд задач:

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.

Индивидуальный вариант:

88. Трекер здоровья Функции — логирование симптомов, отображение истории, генерация отчётов.

TDD — тесты для ввода и анализа симптомов.

ATDD — сценарии для проверки корректности отчётов.

BDD — сценарий «Given симптом записан, When запрашиваю историю, Then данные отображаются корректно».

SDD — таблицы с примерами симптомов и аналитическими данными.

Часть 1. TDD – тесты для ввода и анализа симптомов.

Были созданы модульные тесты в соответствии с методологией TDD:

- TestSymptomsTracker.java - содержит тесты для основных функций:
  - testAddSymptom() - проверка добавления симптома
  - testGetSymptomsHistory() – проверка получения истории симптомов
  - testGenerateReport() – проверка генерации отчета

Реализованы минимальные версии классов с заглушками:

- Symptom.java - класс симптома с базовыми полями и методом isValid(), возвращающим false
- SymptomsTracker.java - класс трекера с методами, возвращающими заглушки:
  - addSymptom() → false
  - getSymptomsHistory() → пустой список
  - generateReport() → null
- HealthReport.java - класс отчета с методами-заглушками

После запуска тестов:

Ожидаемый результат: все тесты должны завершиться неудачно

Фактический результат изображен на рисунке 1.

```
TestSymptomsTracker > testAddSymptom FAILED
    java.lang.AssertionError at TestSymptomsTracker.java:30
TestSymptomsTracker > testGenerateReport FAILED
    java.lang.AssertionError at TestSymptomsTracker.java:54
TestSymptomsTracker > testGetSymptomsHistory FAILED
    java.lang.AssertionError at TestSymptomsTracker.java:42
3 tests completed, 3 failed
```

Рисунок 1 – Результат тестирования

```
> Task :app:compileDebugUnitTestJavaWithJavac
```

Рисунок 2 – Успешная компиляция кода

Код тестирования представлен в листинге 1.

Листинг 1. TestSymptomsTracker.java

```
package com.example.healthtracker;
```

```java
import org.junit.Before;
import org.junit.Test;
import static org.junit.Assert.*;

import java.util.Date;
import java.util.List;

public class TestSymptomsTracker {

    private SymptomsTracker tracker;
    private Symptom testSymptom;

    @Before
    public void setUp() {
        tracker = new SymptomsTracker();
        testSymptom = new Symptom("Головная боль", "Сильная боль в висках",
new Date(), 7);
    }

    @Test
    public void testAddSymptom() {
        // Given
        Symptom symptom = testSymptom;

        // When
        boolean result = tracker.addSymptom(symptom);

        // Then
        assertTrue("Симптом должен быть успешно добавлен", result);
    }

    @Test
    public void testGetSymptomsHistory() {
        // Given
        tracker.addSymptom(testSymptom);

        // When
        List<Symptom> history = tracker.getSymptomsHistory();

        // Then
        assertEquals("История должна содержать 1 симптом", 1,
history.size());
    }

    @Test
    public void testGenerateReport() {
        // Given
        tracker.addSymptom(testSymptom);

        // When
        HealthReport report = tracker.generateReport();

        // Then
        assertNotNull("Отчет не должен быть null", report);
    }
}
```

Часть 2. ATDD — сценарии для проверки корректности отчётов.

Перед реализацией функциональности были согласованы с заказчиком следующие сценарии использования:

Сценарий 1: Добавление симптома

Предусловие: пользователь заходит в раздел «Добавить симптом»

Действия:

1. пользователь вводит название симптома (например, «Головная боль»);
2. указывает описание (например, «Сильная боль в висках»);
3. выбирает интенсивность (например, 7);
4. устанавливает дату (текущая дата);
5. нажимает кнопку «Сохранить».

Ожидаемый результат: симптом «Головная боль» появляется в списке активных симптомов.

Сценарий 2: Генерация отчета о здоровье

Предусловие: пользователь имеет несколько симптомов в истории

Действия:

1. пользователь нажимает кнопку «Создать отчет»;
2. приложение анализирует данные и генерирует отчет.

Ожидаемый результат: система отображает отчет с аналитикой (количество симптомов, средняя интенсивность, самый частый симптом).

Сценарий 3: Валидация вводимых данных

Предусловие: пользователь вводит данные симптома

Действия:

1. пользователь вводит некорректные данные (пустое название, интенсивность вне диапазона);
2. система проверяет валидность данных.

Ожидаемый результат: система отклоняет некорректные данные и сообщает об ошибке.

На основе согласованных сценариев были созданы автоматизированные

приемочные тесты. Тесты представлены в листинге 2.

Листинг 2. TestHealthTrackerAcceptance.java

```
package com.example.healthtracker;

import org.junit.Test;
import static org.junit.Assert.*;

import java.util.Date;

public class TestHealthTrackerAcceptance {

    // Сценарий 1: Добавление симптома
    @Test
    public void testAddSymptom_AcceptanceScenario() {
        // Предусловие: пользователь открыл раздел "Добавить симптом"
        SymptomsTracker tracker = new SymptomsTracker();

        // Действия:
        // 1) пользователь вводит название симптома ("Головная боль")
        // 2) указывает описание ("Сильная боль в висках")
        // 3) выбирает интенсивность (7)
        // 4) устанавливает дату (текущая дата)
        // 5) нажимает кнопку "Сохранить"
        Symptom symptom = new Symptom("Головная боль", "Сильная боль в
висках", new Date(), 7);
        boolean result = tracker.addSymptom(symptom);

        // Ожидаемый результат: симптом появляется в списке активных
симптомов
        assertTrue("Симптом должен быть успешно добавлен", result);
        assertEquals("Симптом должен быть в истории", 1,
tracker.getSymptomsHistory().size());
        assertEquals("Название симптома должно совпадать", "Головная боль",
                tracker.getSymptomsHistory().get(0).getName());
    }

    // Сценарий 2: Генерация отчета о здоровье
    @Test
    public void testGenerateHealthReport_AcceptanceScenario() {
        // Предусловие: пользователь имеет несколько симптомов в истории
        SymptomsTracker tracker = new SymptomsTracker();
        tracker.addSymptom(new Symptom("Головная боль", "Боль в висках", new
Date(), 7));
        tracker.addSymptom(new Symptom("Тошнота", "Легкая тошнота", new
Date(), 4));
        tracker.addSymptom(new Symptom("Усталость", "Сильная усталость", new
Date(), 6));

        // Действия: пользователь нажимает кнопку "Создать отчет"
        HealthReport report = tracker.generateReport();

        // Ожидаемый результат: система генерирует отчет с аналитикой
        assertNotNull("Отчет должен быть создан", report);
        assertEquals("В отчете должно быть 3 симптома", 3,
report.getTotalSymptoms());
        assertTrue("Должна быть рассчитана средняя интенсивность",
                report.getAverageIntensity() > 0);
        assertNotNull("Должен быть определен самый частый симптом",
                report.getMostFrequentSymptom());
    }

    // Сценарий 3: Валидация вводимых данных
```

```
    @Test
    public void testDataValidation_AcceptanceScenario() {
        // Предусловие: пользователь вводит данные симптома
        SymptomsTracker tracker = new SymptomsTracker();

        // Действия: пользователь вводит некорректные данные
        Symptom invalidNameSymptom = new Symptom("", "Описание симптома",
new Date(), 5);
        Symptom invalidIntensitySymptom = new Symptom("Температура",
"Высокая", new Date(), 11);
        Symptom validSymptom = new Symptom("Головная боль", "Обычная боль",
new Date(), 5);

        // Ожидаемый результат: система отклоняет некорректные данные
        assertFalse("Симптом с пустым названием не должен добавляться",
                tracker.addSymptom(invalidNameSymptom));
        assertFalse("Симптом с интенсивностью > 10 не должен добавляться",
                tracker.addSymptom(invalidIntensitySymptom));
        assertTrue("Валидный симптом должен добавляться",
                tracker.addSymptom(validSymptom));
        assertEquals("В истории должен быть только валидный симптом",
                1, tracker.getSymptomsHistory().size());
    }
}
```

Была реализована логика приложения на основе созданных тестов, обеспечив успешное прохождение всех unit-тестов.

Реализация класса Symptom представлена в листинге 2.

- Добавлена валидация в метод isValid()
- Проверка: название не пустое, интенсивность от 1 до 10

Листинг 3. Symptom.java

```
package com.example.healthtracker;

import java.io.Serializable;
import java.util.Date;

public class Symptom implements Serializable {
    private String name;
    private String description;
    private Date date;
    private int intensity;

    public Symptom(String name, String description, Date date, int
intensity) {
        this.name = name;
        this.description = description;
        this.date = date;
        this.intensity = intensity;
    }

    // Геттеры
    public String getName() { return name; }
    public String getDescription() { return description; }
    public Date getDate() { return date; }
    public int getIntensity() { return intensity; }

    // Валидация симптома
    public boolean isValid() {
        return name != null && !name.trim().isEmpty() &&
```

```
                intensity >= 1 && intensity <= 10;
    }
}
```

Реализация класса SymptomsTracker представлена в листинге 3.

- Метод addSymptom(): добавление симптома с проверкой валидности
- Метод getSymptomsHistory(): возврат копии списка симптомов
- Метод generateReport(): генерация отчета с расчетом статистики

Листинг 4. SymptomsTracker.java

```
package com.example.healthtracker;

import java.util.ArrayList;
import java.util.List;

public class SymptomsTracker {
    private List<Symptom> symptomsHistory;

    public SymptomsTracker() {
        this.symptomsHistory = new ArrayList<>();
    }

    public boolean addSymptom(Symptom symptom) {
        if (symptom != null && symptom.isValid()) {
            symptomsHistory.add(symptom);
            return true;
        }
        return false;
    }

    public List<Symptom> getSymptomsHistory() {
        // Возвращаем копию для защиты от внешних изменений
        return new ArrayList<>(symptomsHistory);
    }

    public HealthReport generateReport() {
        HealthReport report = new HealthReport();

        if (symptomsHistory.isEmpty()) {
            return report; // Возвращаем пустой отчет
        }

        // Подсчет общего количества симптомов
        report.setTotalSymptoms(symptomsHistory.size());

        // Расчет средней интенсивности
        double sum = 0;
        for (Symptom symptom : symptomsHistory) {
            sum += symptom.getIntensity();
        }
        report.setAverageIntensity(sum / symptomsHistory.size());

        return report;
    }
}
```

Реализация класса HealthReport представлена в листинге 4.

- Добавлены поля для хранения аналитических данных
- Реализованы геттеры и сеттеры для доступа к статистике

Листинг 5. HealthReport.java

```
package com.example.healthtracker;

import java.util.ArrayList;
import java.util.List;

public class HealthReport {
    private int totalSymptoms;
    private double averageIntensity;
    private List<String> frequentSymptoms;

    public HealthReport() {
        this.totalSymptoms = 0;
        this.averageIntensity = 0.0;
        this.frequentSymptoms = new ArrayList<>();
    }

    // Геттеры и сеттеры
    public int getTotalSymptoms() { return totalSymptoms; }
    public void setTotalSymptoms(int totalSymptoms) { this.totalSymptoms =
totalSymptoms; }

    public double getAverageIntensity() { return averageIntensity; }
    public void setAverageIntensity(double averageIntensity) {
        this.averageIntensity = averageIntensity;
    }

    public List<String> getFrequentSymptoms() { return frequentSymptoms; }
    public void setFrequentSymptoms(List<String> frequentSymptoms) {
        this.frequentSymptoms = frequentSymptoms;
    }
}
```

После запуска тестов:

Ожидаемый результат: все unit-тесты прошли успешно после реализации логики.

Фактический результат изображен на рисунке 3.

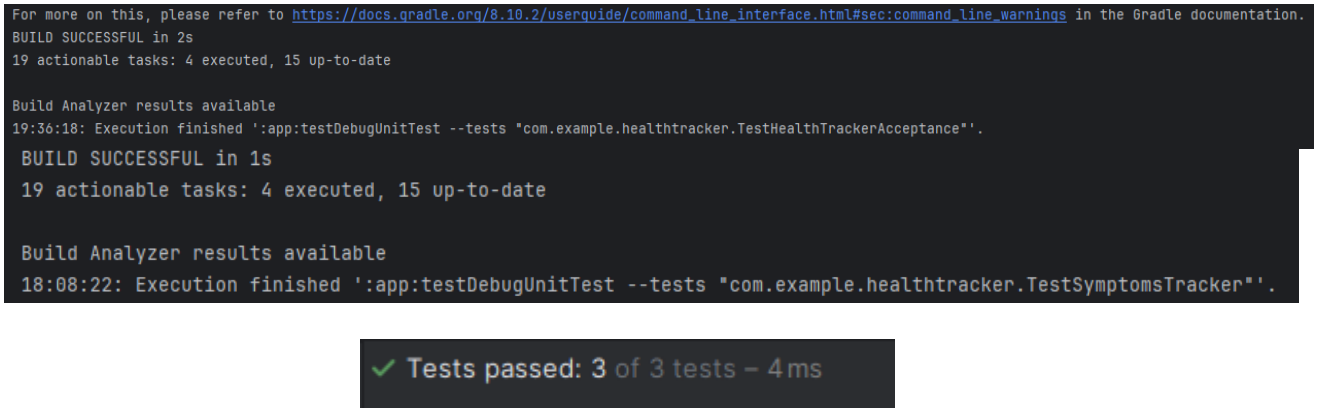

Рисунок 3 – Результат тестирования

Бизнес-логика приложения успешно реализована. Все созданные на первом этапе тесты проходят, что подтверждает: корректность реализации требований к функциональности, работоспособность TDD

процесса, готовность системы к следующему этапу разработки

Часть 3. BDD — сценарий «Given, When, Then»

В соответствии с методологией BDD были сформулированы сценарии на языке Gherkin, понятном для всех участников проекта. Сценарии представлены в листинге 6.

Листинг 6. health_tracker. Feature

```
Функция: Трекер здоровья
  Как пользователь, заботящийся о своем здоровье
  Я хочу отслеживать свои симптомы
  Чтобы анализировать состояние своего здоровья

  Сценарий: Пользователь добавляет симптом и проверяет историю
    Дано у меня есть трекер симптомов
    Когда я добавляю симптом с названием "Головная боль" описанием "Сильная
боль в висках" и интенсивностью 7
    Тогда симптом "Головная боль" должен быть в истории симптомов

  Сценарий: Пользователь просматривает историю нескольких симптомов
    Дано я добавил несколько симптомов в трекер
    Когда я запрашиваю историю симптомов
    Тогда я должен увидеть все добавленные симптомы
    И история должна содержать 2 симптомов

  Сценарий: Пользователь генерирует отчет о здоровье
    Дано я добавил несколько симптомов в трекер
    Когда я генерирую отчет о здоровье
    Тогда отчет должен содержать общее количество симптомов 2
    И отчет должен содержать среднюю интенсивность

  Сценарий: Валидация некорректных данных симптома
    Дано у меня есть трекер симптомов
    Когда я добавляю симптом с названием "" описанием "Пустое название" и
интенсивностью 5
    Тогда симптом не должен быть добавлен в историю
```

Таблица 1. Использование сценария GWT

Шаг BDD	Описание	Код реализации
Given	Симптом записан	tracker.addSymptom(symptom)
When	Запрашиваю историю	tracker.getSymptomsHistory()
Then	Данные отображаются корректно	Проверка полей симптома

Для автоматизации BDD-сценариев был создан класс HealthTrackerSteps, который связывает текстовые шаги с кодом приложения. Реализация класса представлена в листинге 7.

Листинг 7. HealthTrackerSteps.java

```
package com.example.healthtracker;

import io.cucumber.java.en.Given;
import io.cucumber.java.en.When;
```

```java
import io.cucumber.java.en.Then;
import static org.junit.Assert.*;

import java.util.Date;
import java.util.List;

public class HealthTrackerSteps {

    private SymptomsTracker tracker;
    private HealthReport report;
    private boolean addResult;
    private List<Symptom> history;

    @Given("у меня есть трекер симптомов")
    public void у_меня_есть_трекер_симптомов() {
        tracker = new SymptomsTracker();
    }

    @When("я добавляю симптом с названием {string} описанием {string} и
интенсивностью {int}")
    public void
я_добавляю_симптом_с_названием_описанием_и_интенсивностью(String name,
String description, int intensity) {
        Symptom symptom = new Symptom(name, description, new Date(),
intensity);
        addResult = tracker.addSymptom(symptom);
    }

    @Then("симптом {string} должен быть в истории симптомов")
    public void симптом_должен_быть_в_истории_симптомов(String expectedName)
{
        assertTrue("Симптом должен быть успешно добавлен", addResult);
        List<Symptom> history = tracker.getSymptomsHistory();
        assertFalse("История не должна быть пустой", history.isEmpty());
        assertEquals("Название симптома должно совпадать", expectedName,
history.get(0).getName());
    }

    @Given("я добавил несколько симптомов в трекер")
    public void я_добавил_несколько_симптомов_в_трекер() {
        tracker = new SymptomsTracker();
        tracker.addSymptom(new Symptom("Головная боль", "Боль в висках", new
Date(), 7));
        tracker.addSymptom(new Symptom("Тошнота", "Легкая тошнота", new
Date(), 4));
    }

    @When("я запрашиваю историю симптомов")
    public void я_запрашиваю_историю_симптомов() {
        history = tracker.getSymptomsHistory();
    }

    @Then("я должен увидеть все добавленные симптомы")
    public void я_должен_увидеть_все_добавленные_симптомы() {
        assertNotNull("История не должна быть null", history);
        assertEquals("Должно быть 2 симптома в истории", 2, history.size());
    }

    @Then("история должна содержать {int} симптомов")
    public void история_должна_содержать_симптомов(int expectedCount) {
        assertEquals("Количество симптомов должно совпадать", expectedCount,
history.size());
    }
```

```
    @Then("симптом не должен быть добавлен в историю")
    public void симптом_не_должен_быть_добавлен_в_историю() {
        assertFalse("Симптом не должен быть добавлен", addResult);
        List<Symptom> history = tracker.getSymptomsHistory();
        assertTrue("История должна быть пустой", history.isEmpty());
    }

    // Дополнительные шаги для генерации отчетов
    @When("я генерирую отчет о здоровье")
    public void я_генерирую_отчет_о_здоровье() {
        report = tracker.generateReport();
    }

    @Then("отчет должен содержать общее количество симптомов {int}")
    public void отчет_должен_содержать_общее_количество_симптомов(int
expectedTotal) {
        assertNotNull("Отчет не должен быть null", report);
        assertEquals("Общее количество симптомов должно совпадать",
expectedTotal, report.getTotalSymptoms());
    }

    @Then("отчет должен содержать среднюю интенсивность")
    public void отчет_должен_содержать_среднюю_интенсивность() {
        assertNotNull("Отчет не должен быть null", report);
        assertTrue("Средняя интенсивность должна быть рассчитана",
report.getAverageIntensity() >= 0);
    }
}
```

Также создан RunCucumberTest класс. Реализация представлена в листинге 8.

Листинг 8. RunCucumberTest.java

```
package com.example.healthtracker;

import org.junit.runner.RunWith;
import io.cucumber.junit.Cucumber;
import io.cucumber.junit.CucumberOptions;

@RunWith(Cucumber.class)
@CucumberOptions(
        features = "src/test/resources/com/example/healthtracker",
        glue = "com.example.healthtracker",
        plugin = {"pretty", "html:target/cucumber-reports"}
)
public class RunCucumberTest {
}
```

Проведен рефакторинг кода для улучшения его качества и создан полноценный пользовательский интерфейс для взаимодействия с приложением. Улучшения в классе SymptomsTracker.java представлены в листинге 9.

Листинг 9. SymptomsTracker.java

```
// Добавлены новые методы для улучшения функциональности
public List<Symptom> getSortedSymptomsHistory() {
    return symptomsHistory.stream()
            .sorted(Comparator.comparing(Symptom::getDate).reversed())
            .collect(Collectors.toList());
}
```

```
private double calculateAverageIntensity() {
    return symptomsHistory.stream()
            .mapToInt(Symptom::getIntensity)
            .average()
            .orElse(0.0);
}

private String findMostFrequentSymptom() {
    return symptomsHistory.stream()
            .collect(Collectors.groupingBy(Symptom::getName,
Collectors.counting()))
            .entrySet().stream()
            .max(Map.Entry.comparingByValue())
            .map(Map.Entry::getKey)
            .orElse("Нет данных");
}
```

Улучшения в классе HealthReport.java представлены в листинге 10.

- Добавлен метод getFormattedReport() для красивого форматирования отчета
- Реализовано хранение наиболее частого симптома
- Добавлены сеттеры для всех полей отчета

Листинг 10. HealthReport.java

```
// Сеттеры
public void setTotalSymptoms(int totalSymptoms) { this.totalSymptoms =
totalSymptoms; }
public void setAverageIntensity(double averageIntensity) {
    this.averageIntensity = averageIntensity;
}
public void setMostFrequentSymptom(String mostFrequentSymptom) {
    this.mostFrequentSymptom = mostFrequentSymptom;
}
public void setFrequentSymptoms(List<String> frequentSymptoms) {
    this.frequentSymptoms = frequentSymptoms;
}


public String getFormattedReport() {
    DecimalFormat df = new DecimalFormat("#.##");
    return String.format(
            "Отчет о здоровье:\n" +
                    "Всего симптомов: %d\n" +
                    "Средняя интенсивность: %s\n" +
                    "Самый частый симптом: %s",
            totalSymptoms,
            df.format(averageIntensity),
            mostFrequentSymptom
    );
}
```

Был создан пользовательский интерфейс:

Главный экран (activity_main.xml**):**

- Кнопка "Добавить симптом" для перехода к форме ввода
- Кнопка "История симптомов" для просмотра сохраненных данных

- Кнопка "Создать отчет" для генерации аналитики
- Панель статистики с количеством симптомов
- Область для отображения отчетов и истории

Экран добавления симптома (activity_add_symptom.xml**):**

- Поле ввода названия симптома с валидацией
- Многострочное поле для описания
- SeekBar для выбора интенсивности (1–10)
- Индикатор текущего значения интенсивности
- Кнопки "Сохранить" и "Отмена"

Создан класс AddSymptomActivity.java, который позволил обрабатывать ввод данных пользователем, передавать данные обратно в главную активность. Появление визуальной обратной связи через изменение цвета индикатора интенсивности, валидация обязательных полей. Логика работы представлена в листинге 11.

Листинг 11. Логика работы AddSymptomActivity.java

```
private void saveSymptom() {
    String name = etSymptomName.getText().toString().trim();
    String description = etDescription.getText().toString().trim();

    if (name.isEmpty()) {
        etSymptomName.setError("Введите название симптома");
        etSymptomName.requestFocus();
        return;
    }

    Symptom symptom = new Symptom(name, description, new Date(),
currentIntensity);

    Intent resultIntent = new Intent();
    resultIntent.putExtra("symptom", symptom);
    setResult(Activity.RESULT_OK, resultIntent);

    Toast.makeText(this, "Симптом сохранен!", Toast.LENGTH_SHORT).show();
    finish();
}
```

Обновлен MainActivity.java:

- Реализована обработка результата из AddSymptomActivity
- Добавлен метод onActivityResult() для получения данных
- Улучшено отображение истории симптомов
- Реализовано обновление UI при добавлении новых данных

Листинг 12. MainActivity.java

```
package com.example.healthtracker;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;
import androidx.appcompat.app.AppCompatActivity;

import java.util.List;

public class MainActivity extends AppCompatActivity {

    private SymptomsTracker symptomsTracker;
    private TextView tvTotalSymptoms, tvReport;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        // Инициализация трекера
        symptomsTracker = new SymptomsTracker();

        // Инициализация View
        initializeViews();
        setupClickListeners();
        updateUI();
    }

    private void initializeViews() {
        tvTotalSymptoms = findViewById(R.id.tvTotalSymptoms);
        tvReport = findViewById(R.id.tvReport);
    }

    private void setupClickListeners() {
        Button btnAddSymptom = findViewById(R.id.btnAddSymptom);
        Button btnViewHistory = findViewById(R.id.btnViewHistory);
        Button btnGenerateReport = findViewById(R.id.btnGenerateReport);

        btnAddSymptom.setOnClickListener(v -> openAddSymptomActivity());
        btnViewHistory.setOnClickListener(v -> showHistory());
        btnGenerateReport.setOnClickListener(v -> generateAndShowReport());
    }

    private void openAddSymptomActivity() {
        Intent intent = new Intent(MainActivity.this,
AddSymptomActivity.class);
        startActivityForResult(intent, 1); // 1 - request code
    }

    @Override
    protected void onActivityResult(int requestCode, int resultCode, Intent
data) {
        super.onActivityResult(requestCode, resultCode, data);

        if (requestCode == 1 && resultCode == Activity.RESULT_OK) {
            if (data != null && data.hasExtra("symptom")) {
                Symptom symptom = (Symptom)
data.getSerializableExtra("symptom");
                if (symptom != null && symptomsTracker.addSymptom(symptom))
{
                    Toast.makeText(this, "Симптом успешно добавлен!",
```

```
Toast.LENGTH_SHORT).show();
                    updateUI();
                } else {
                    Toast.makeText(this, "Ошибка при добавлении симптома",
Toast.LENGTH_SHORT).show();
                }
            }
        }
    }

    private void showHistory() {
        List<Symptom> history = symptomsTracker.getSymptomsHistory();
        StringBuilder historyText = new StringBuilder("История
симптомов:\n");

        if (history.isEmpty()) {
            historyText.append("История симптомов пуста");
        } else {
            for (Symptom symptom : history) {
                historyText.append(String.format(
                        "• %s (интенсивность: %d)\n  %s\n\n",
                        symptom.getName(),
                        symptom.getIntensity(),
                        symptom.getDescription()
                ));
            }
        }

        tvReport.setText(historyText.toString());
    }

    private void generateAndShowReport() {
        HealthReport report = symptomsTracker.generateReport();
        tvReport.setText(report.getFormattedReport());
    }

    private void updateUI() {

tvTotalSymptoms.setText(String.valueOf(symptomsTracker.getSymptomsCount()));
    }
}
```

Рефакторинг и создание пользовательского интерфейса успешно завершены. Приложение превратилось из набора классов с бизнес-логикой в полноценное Android-приложение. Демонстрация работы пользовательского интерфейса изображено на рисунках 4–7.

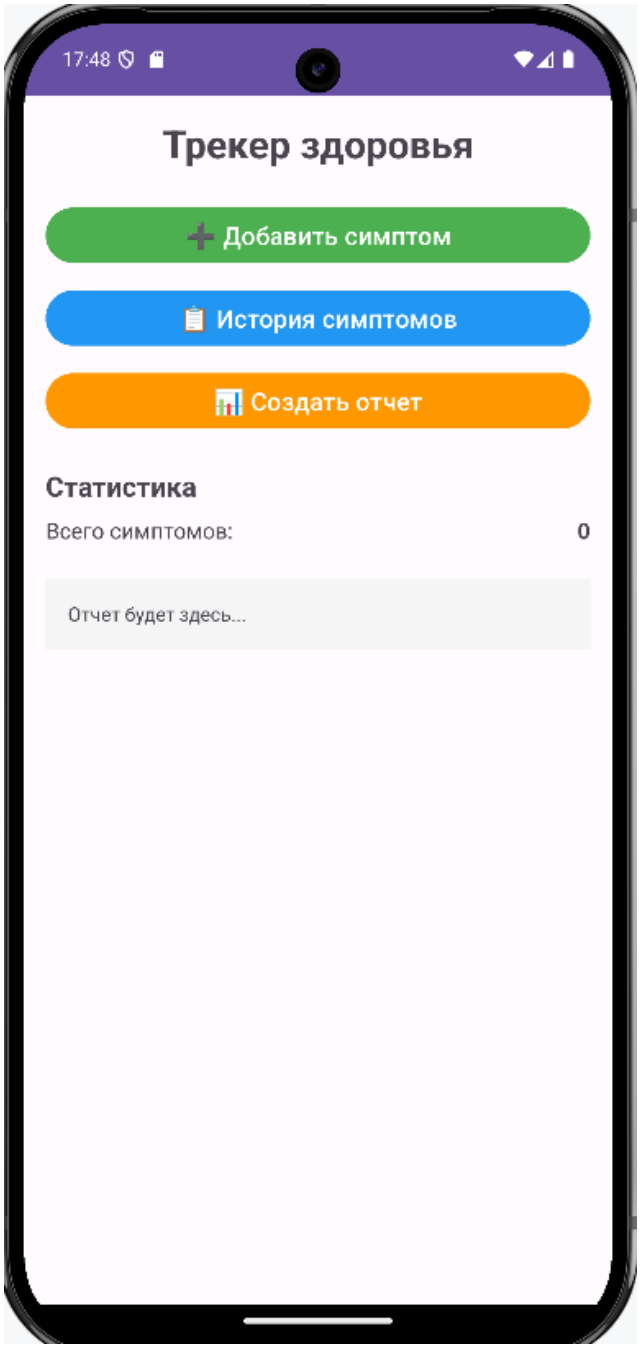

Рисунок 4 – Главная активность

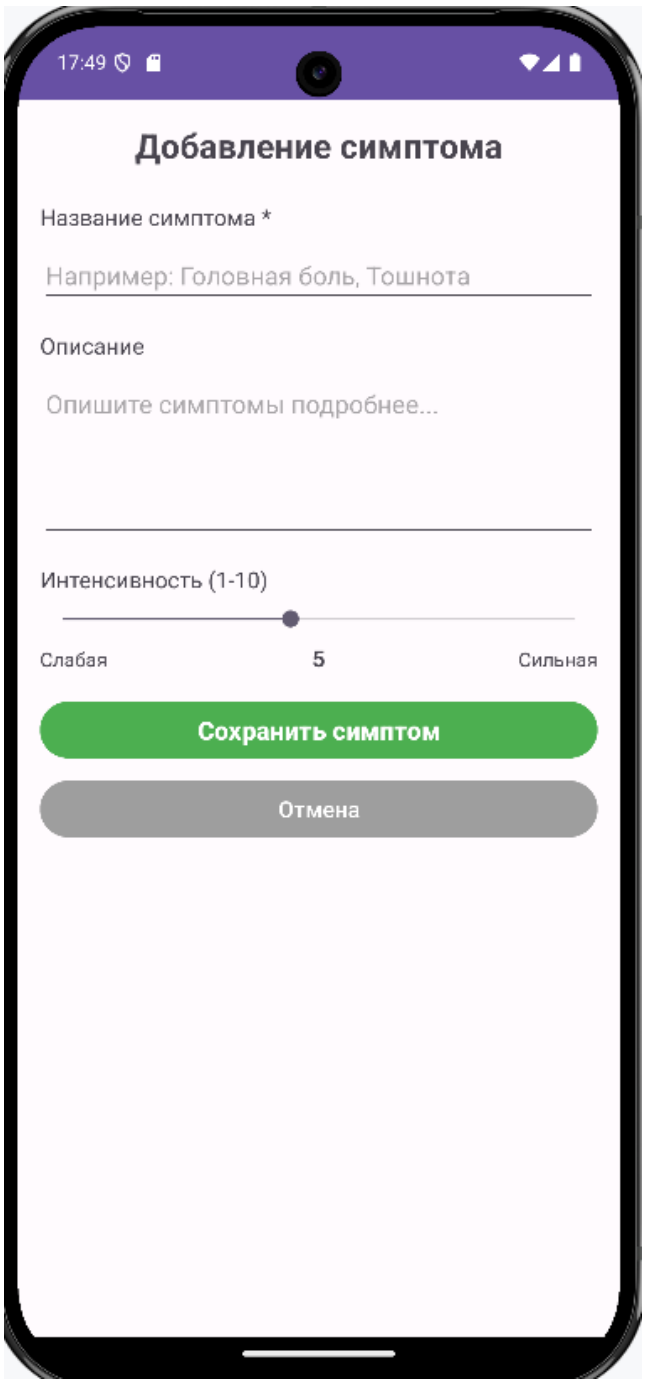

Рисунок 5 – Активность добавления симптома

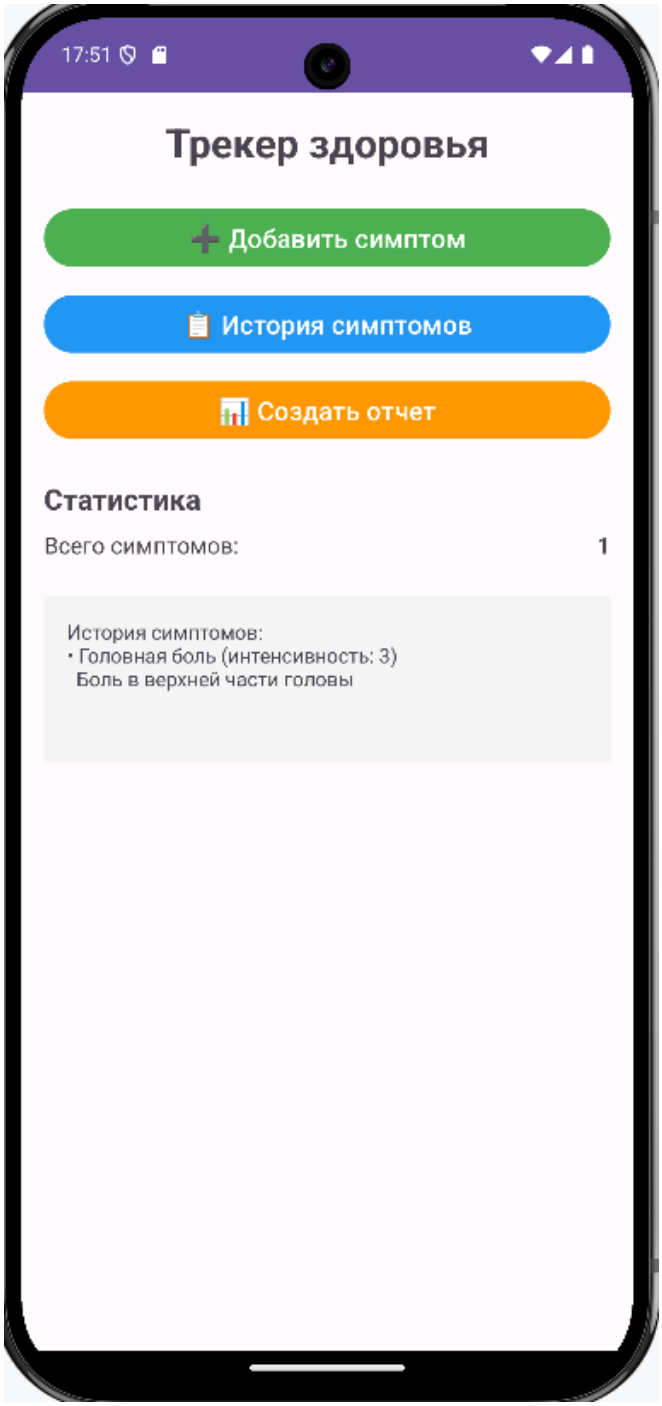

Рисунок 6 – Главная активность после добавления симптома

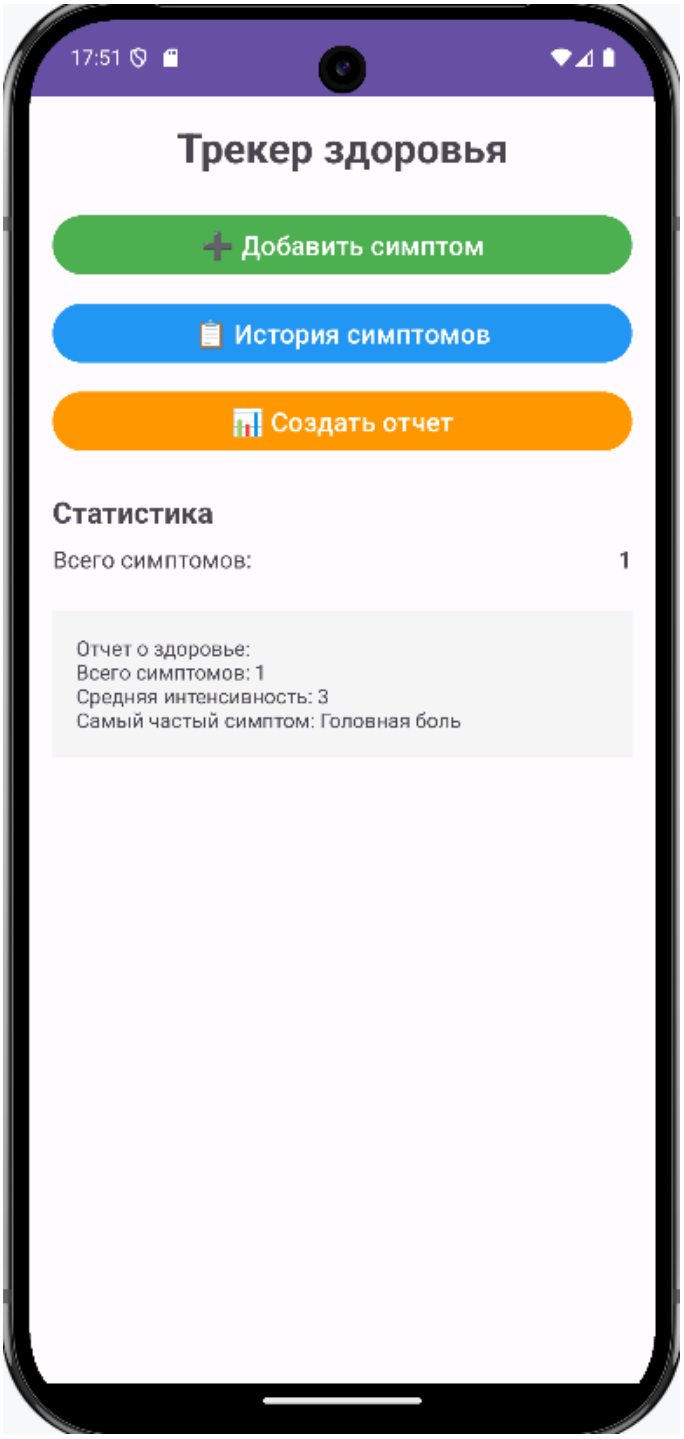

Рисунок 7 – Создание отчета

Часть 4. SDD — таблицы с примерами симптомов и аналитическими данными.

Таблица 2. Примеры симптомов для тестирования

Название симптома	Описание	Интенсивность	Валидность	Комментарий
Головная боль	Боль в висках	7	Да	Корректные данные
Тошнота	После еды	4	Да	Нормальная интенсивность
Усталость	Вечерняя усталость	6	Да	Стандартный случай
Пустая строка	Описание	5	Нет	Отсутствует название
Температура	37.5°C	11	Нет	Интенсивность вне диапазона

Таблица 3. Аналитические данные

Метрика	Формула	Пример	Результат
Количество симптомов	COUNT (симптомы)	[Головная боль, Тошнота, Усталость]	3
Самый частый симптом	MODE (названия)	[А, Б, А]	"А"
Средняя интенсивность	Σ интенсивность / количество	[7,4,6]	5.67

Таблица 4. Сценарии генерации отчетов

Исходные данные	Действие	Ожидаемый результат
Пустой трекер	Генерация отчета	Отчет: 0 симптомов, средняя интенсивность 0
1 симптом: Головная боль (интенсивность 7)	Генерация отчета	Отчет: 1 симптом, средняя интенсивность 7.0
3 симптома: [7, 4, 6]	Генерация отчета	Отчет: 3 симптома, средняя интенсивность 5,67

На основе составленных спецификаций были созданы автоматизированные тесты. Реализация тестов представлена в листингах 13–14.

Листинг 13. TestHealthTrackerSDD.java

```
package com.example.healthtracker;

import org.junit.Test;
import org.junit.runner.RunWith;
import org.junit.runners.Parameterized;
import static org.junit.Assert.*;

import java.util.Arrays;
import java.util.Collection;
import java.util.Date;

@RunWith(Parameterized.class)
public class TestHealthTrackerSDD {

    private String symptomName;
    private String description;
    private int intensity;
    private boolean expectedValidity;

    public TestHealthTrackerSDD(String symptomName, String description,
                                int intensity, boolean expectedValidity) {
        this.symptomName = symptomName;
        this.description = description;
        this.intensity = intensity;
        this.expectedValidity = expectedValidity;
    }

    @Parameterized.Parameters
    public static Collection<Object[]> testData() {
        return Arrays.asList(new Object[][] {
                // Данные из Таблицы 1
                {"Головная боль", "Сильная боль в висках", 7, true},
                {"Тошнота", "После еды", 4, true},
                {"Усталость", "Вечерняя усталость", 6, true},
                {"", "Описание симптома", 5, false},
                {"Температура", "37.5°C", 11, false},
                {"Кашель", "Сухой кашель", 0, false}
        });
    }

    @Test
    public void testSymptomValidationBasedOnExamples() {
        // Given
        Symptom symptom = new Symptom(symptomName, description, new Date(),
intensity);

        // When
        boolean isValid = symptom.isValid();

        // Then
        assertEquals("Валидация симптома '" + symptomName + "' с
интенсивностью " + intensity,
                expectedValidity, isValid);
    }
}
```

Результат запуска тестов представлен на рисунке 8.

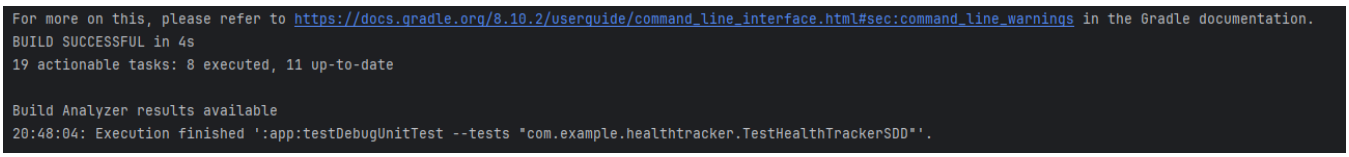

Рисунок 8 – Запуск теста

Листинг 14. TestHealthTrackerAnalyticsSDD.java

```
package com.example.healthtracker;

import org.junit.Test;
import static org.junit.Assert.*;

import java.util.Date;

public class TestHealthTrackerAnalyticsSDD {

    @Test
    public void testReportGenerationWithThreeSymptoms() {
        // Given - данные из Таблицы 2
        SymptomsTracker tracker = new SymptomsTracker();
        tracker.addSymptom(new Symptom("Головная боль", "Боль в висках", new
Date(), 7));
        tracker.addSymptom(new Symptom("Тошнота", "После еды", new Date(),
4));
        tracker.addSymptom(new Symptom("Усталость", "Вечерняя усталость",
new Date(), 6));

        // When
        HealthReport report = tracker.generateReport();

        // Then - проверка ожидаемых результатов из Таблицы 2
        assertEquals("Количество симптомов должно быть 3", 3,
report.getTotalSymptoms());
        assertEquals("Средняя интенсивность должна быть 5.67", 5.67,
report.getAverageIntensity(), 0.01);
    }

    @Test
    public void testEmptyTrackerReport() {
        // Given - пустой трекер (данные из Таблицы 3)
        SymptomsTracker tracker = new SymptomsTracker();

        // When
        HealthReport report = tracker.generateReport();

        // Then
        assertEquals("Отчет должен быть пустым", 0,
report.getTotalSymptoms());
        assertEquals("Средняя интенсивность должна быть 0", 0.0,
report.getAverageIntensity(), 0.01);
    }

    @Test
    public void testSingleSymptomReport() {
        // Given - 1 симптом (данные из Таблицы 3)
        SymptomsTracker tracker = new SymptomsTracker();
        tracker.addSymptom(new Symptom("Головная боль", "Сильная боль", new
Date(), 7));

        // When
        HealthReport report = tracker.generateReport();
```

```
        // Then
        assertEquals("Должен быть 1 симптом", 1, report.getTotalSymptoms());
        assertEquals("Средняя интенсивность должна быть 7.0", 7.0,
report.getAverageIntensity(), 0.01);
    }
}
```

Результат запуска тестов представлен на рисунке 9.

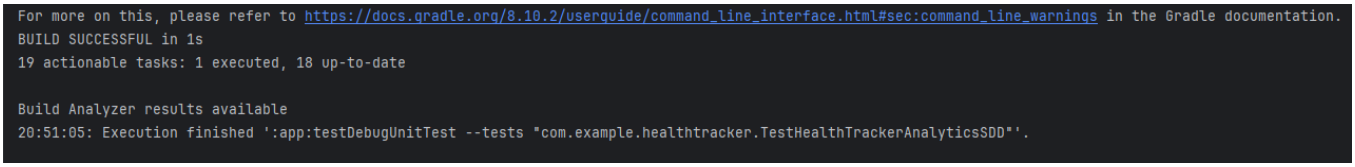

Рисунок 9 – Запуск теста

Вывод

В ходе выполнения практической работы был успешно разработан и протестирован "Трекер здоровья" с применением четырех методологий разработки через тестирование. Реализация началась с подходом TDD, где первоначально были созданы модульные тесты для основных функций приложения, которые закономерно завершились неудачей на этапе "красных тестов". После этого была реализована минимальная функциональность классов Symptom, SymptomsTracker и HealthReport, что позволило успешно пройти все тесты на этапе "зеленых тестов" и подтвердить корректность базовой бизнес-логики.

На этапе ATDD были определены и автоматизированы приемочные тесты, основанные на пользовательских сценариях, что обеспечило соответствие системы ожиданиям заказчика.

Подход BDD позволил формализовать требования через читаемые сценарии на языке Gherkin, улучшив коммуникацию между всеми участниками проекта и создав живую документацию.

Методология SDD продемонстрировала свою эффективность через использование конкретных примеров в виде таблиц с тестовыми данными, что устранило неоднозначности в требованиях и обеспечило полное покрытие тестами критических сценариев использования.

Интеграция всех четырех подходов позволила создать надежное и качественное приложение с хорошо протестированной архитектурой. Разработанный трекер здоровья успешно выполняет все поставленные задачи: логирование симптомов с валидацией данных, отображение истории в хронологическом порядке и генерацию аналитических отчетов с расчетом ключевых метрик. Практическая работа продемонстрировала, что сочетание TDD, ATDD, BDD и SDD значительно повышает качество программного обеспечения, снижает количество ошибок и улучшает процесс коммуникации в команде разработки.

Список используемых источников

1. МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО КУРСОВОМУ ПРОЕКТИРОВАНИЮ, 2024.– режим доступа: URL: https://online edu.mirea.ru/course/view.php?id=7400, свободный (дата обращения 19.03.2024)

2. GITHUB glide, 2016. режим доступа: https://github.com/bumptech/glide, свободный (дата обращения 20.05.2024)

3. Firebase Documentation, 2022. URL: – режим доступа: URL: The https://firebase.google.com/docs?hl=en, свободный (дата обращения 09.05.2024)

4. Movie режим доступа: свободный (дата обращения 05.05.2024)

5. Database (TMDB), 2024. URL: https://www.themoviedb.org/, IMDb, 2022. – режим доступа: URL: https://www.imdb.com/, свободный (дата обращения 13.04.2024)