МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

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»

Выполнил студент группы *ИКБО-74-23*

Кавказский И.К.

Принял Ильичев Г.П.

Практическая работа выполнена «31» октября 2025 г. _____

«Зачтено» «__»_____2025 г. _____

Москва 2025

1. Введение.

Цель работы: Изучение и практическое применение подходов к разработке программного обеспечения, основанных на тестировании (TDD, ATDD, BDD, SDD), для повышения качества, надёжности и поддерживаемости кода.

Задачи работы:

1. Изучить теоретические основы методологий TDD, ATDD, BDD и SDD.
2. Реализовать практический пример для каждого метода.
3. Проанализировать влияние интеграции тестирования на архитектуру и качество программного продукта.
4. Подготовить итоговый отчёт с выводами по проделанной работе.

Вариант задания: 46

Название: Приложение для прогноза погоды с оповещениями.

Функции: получение прогноза, установка оповещений, уведомление при изменениях.

2. Теоретический раздел

- Test-Driven Development (TDD) — Разработка через тестирование. Это методология, при которой тесты для новой функциональности пишутся до написания самого кода. Разработка ведётся короткими циклами «Красный» (тест не проходит) → «Зелёный» (пишется минимальный код для прохождения теста) → «Рефакторинг» (код улучшается без изменения функциональности).

- Acceptance Test-Driven Development (ATDD) — Разработка через приёмочные тесты. Этот подход расширяет TDD, фокусируясь на требованиях конечного пользователя. Вся команда (заказчик, аналитики, разработчики, тестировщики) совместно определяет критерии приёмки в виде тестов, которые описывают, как система должна работать с точки зрения бизнеса.

- Behavior-Driven Development (BDD) — Разработка через поведение. BDD является развитием TDD и ATDD. Основная идея — описывать поведение системы на естественном, понятном для всех языке с использованием структуры Given-When-Then (Дано-Когда-Тогда). Эти описания служат одновременно и документацией, и основой для автоматизированных тестов.

- Specification by Example (SDD) — Спецификация на примерах. Этот подход использует конкретные, реальные примеры для формулирования требований. Вместо абстрактных правил создаются таблицы с входными данными и ожидаемыми результатами, что устраняет двусмысленность и служит живой документацией и основой для тестов.

3. Практическая часть

3.1. Этап 1: Реализация с помощью TDD (Test-Driven Development)

Разработка началась с создания теста, описывающего ожидаемое поведение функции получения прогноза погоды. Тест test_get_forecast проверяет, что в результате вызова функции get_forecast возвращается правильная строка (для проверки правильности строки была реализована функция check_forecast_output) или сообщение о том что город не найден.

test_weather.py > ...

```python
from main import send_notifications_real_true_func, enable_notifications, disable_notifications
from weather_api import get_forecast
import os
from db import set_temp, set_client
#Сосиска

apikey = os.getenv("WEATHER_API_KEY")

def check_forecast_output(city: str, days: int):
    text = get_forecast(city, days)
    lines = [line.strip() for line in text.strip().split('\n') if line.strip()]

    if not lines:
        return False

    day_blocks = []
    current_block = []

    for line in lines:
        if line.startswith('*Дата:') and line.endswith('*'):
            if current_block:
                day_blocks.append(current_block)
            current_block = [line]
        elif current_block:
            current_block.append(line)

    if current_block:
        day_blocks.append(current_block)

    days_count = len(day_blocks)

    if days_count != days:
        return False

    if not (1 <= days_count <= 14):
        return False

    for day_block in day_blocks:
        if len(day_block) != 4:
            return False

        date_line = day_block[0]
        if not (date_line.startswith('*Дата: ') and date_line.endswith('*')):
            return False

        temp_line = day_block[1]
        if not temp_line.startswith('├Средняя температура: ') or not temp_line.endswith('°C'):
            return False

        temp_str = temp_line.replace('├Средняя температура: ', '').replace('°C', '')
        try:
            temp = float(temp_str)
            if not (-50 <= temp <= 50):
                return False
        except (ValueError, TypeError):
            return False

        weather_line = day_block[2]
        if not weather_line.startswith('├Погода: '):
            return False

        precip_line = day_block[3]
        if not precip_line.startswith('└Вероятность осадков: ') or not precip_line.endswith('%'):
            return False

        precip_str = precip_line.replace('└Вероятность осадков: ', '').replace('%', '')
        try:
            precip = float(precip_str)
            if not (0 <= precip <= 100):
                return False
        except (ValueError, TypeError):
            return False

    return True


def test_get_forecast():
    assert check_forecast_output("Москва", 14)
    assert check_forecast_output("Париж, Россия", 5)
    assert get_forecast("Арстоцка", 8) == "Город не найден."
```


Рис. 1, 2 – Тест test_get_forecast

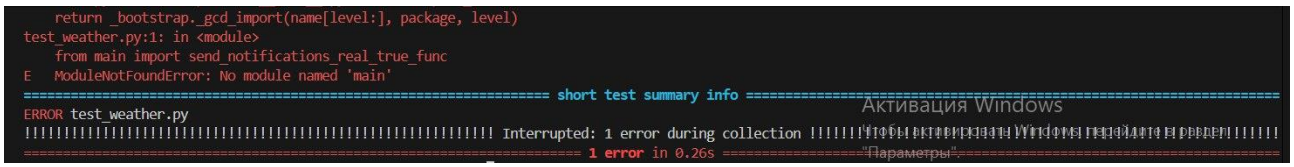


Рис. 3 – Результат теста

Запуск тестов приводил к ожидаемому провалу, так как проверяемых функций не существует

3.1.2 Реализация для прохождения теста

Далее был написан код для того, чтобы тест test_get_forecast успешно прошёл.

```
def get_forecast(city: str, days: int):
    res = requests.get(
        f"http://api.weatherapi.com/v1/forecast.json?key={apikey}&q={city}&days={days}"
    )
    data = res.json()
    if data.get("error"):
        error = data.get("error").get("message", "Неизвестная ошибка")
        return error_messages.get(error, "Неизвестная ошибка")

    forecast = data.get("forecast")
    forecastday = forecast.get("forecastday")
    dayslist = []

    for i in range(len(forecastday)):
        forecastdayinfo = (forecastday[i].get("day"))
        condition_text = forecastdayinfo.get("condition").get("text").lower()
        condition = condition_names.get(condition_text, condition_text)
        chance_of_precipitation = round(((1-((1-(int(forecastdayinfo.get("daily_chance_of_rain"))/100)) * (1-(int(forecastdayinfo.get("daily_chance_of_snow"))/100))))
        forecastdate = str(forecastday[i].get("date"))
        year, month, day = map(int, forecastdate.split('-'))
        date_obj = datetime.date(year, month, day)
        week_day = weekdays[date_obj.weekday()]

        dayslist.append("\n*Дата: "
                        + week_day
                        + ", "
                        + forecastdate
                        + "*\nСредняя температура: "
                        + str(forecastdayinfo.get("avgtemp_c"))
                        + "°C\nПогода: "
                        + condition
                        + "\nВероятность осадков: "
                        + str(chance_of_precipitation) + "%")

    return(
        f"Предполагаемая погода на следующие {days} {day_endings.get(days)}: "
        + "\n".join(dayslist)
    )
```


Рис. 4 – Минимальная реализация get_forecast

После этого изменения тест был запущен повторно и успешно пройден.

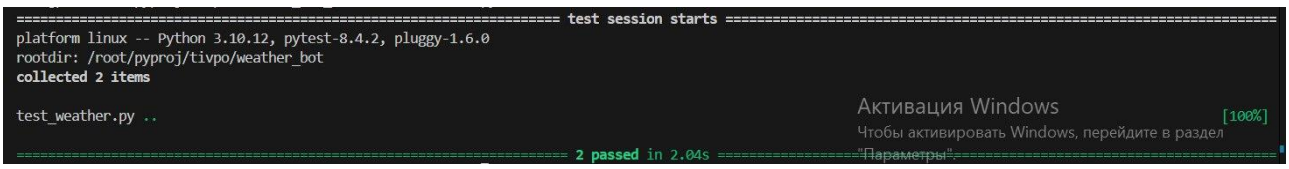


Рис. 5 – Результат теста

3.1.3 Реализация функции удаления заметки (test_notifications)

Был написан новый тест test_notifications. Он проверяет функции включения и отключения уведомлений.

```python

def test_notifications():
    set_client(123, "Брянск")
    disable_notifications(123)
    set_temp("Брянск", 500)
    pinged_users, unpinged_users = send_notifications_real_true_func()
    assert 123 in unpinged_users
    enable_notifications(123)
    set_temp("Брянск", 500)
    pinged_users, unpinged_users = send_notifications_real_true_func()
    assert 123 in pinged_users
```

Рис. 6 – Тест test_notifications

На момент написания теста функции send_notifications_real_true_func, disable_notifications, enable_notifications и другие отсутствовали в коде, поэтому запуск теста приводил к ошибке.

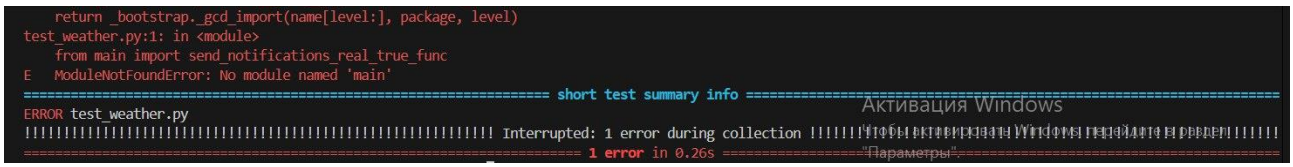


Рис. 7 – Результат теста

3.1.5 Реализация функций

Далее был написан код нужных функций:

- **init_database** для создания бд
- **set_client** для добавления клиента в базу данных
- **enable_notifications** для включения уведомлений
- **disable_notifications** для отключения уведомлений
- **change_notification_settings** для смены настроек в бд
- **dict_to_string** и **string_to_dict** для конвертации словаря в сохраняемую в бд строку и наоборот
- **send_notifications_real_true_func** для отправки уведомлений
- **get_all_notification_settings** для извлечения настроек из бд
- **check_notifications** для проверки кого из пользователей нужно оповещать
- **get_cities** и **get_temp** для получения списка городов и температур в них из бд
- **get_current_temp** для получения текущей погоды в городе

```python
def init_database():
    conn = sqlite3.connect('weather.db')
    cursor = conn.cursor()

    cursor.execute('''
    CREATE TABLE IF NOT EXISTS clients (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        tg_id INTEGER,
        city TEXT
    )
    ''')

    cursor.execute('''
    CREATE TABLE IF NOT EXISTS temperatures (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        city TEXT UNIQUE,
        temperature FLOAT
    )
    ''')

    cursor.execute('''
    CREATE TABLE IF NOT EXISTS notify_settings (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        tg_id INTEGER,
        should_notify INTEGER,
        settings TEXT
    )
    ''')

    conn.commit()
    conn.close()

    print("База данных успешно инициализирована!")
```

Рис. 8 – Функция init_database

```python
def set_client(tg_id: int, city: str):
    conn = sqlite3.connect('weather.db')
    cursor = conn.cursor()

    city_temp = get_current_temp(city)
    client = (tg_id, city)
    city_data = (city, city_temp)

    conn.execute(f"DELETE FROM clients WHERE tg_id = {tg_id}")
    conn.commit()

    cursor.execute('''
    INSERT INTO clients (tg_id, city)
    VALUES (?, ?)
    ''', client)

    if city_temp != "None":
        cursor.execute('''
        INSERT OR IGNORE INTO temperatures (city, temperature)
        VALUES (?, ?)
        ''', city_data)

    data = conn.execute(f"SELECT * FROM notify_settings n WHERE n.tg_id = {tg_id}").fetchone()
    if data is None:
        conn.execute(f"INSERT INTO notify_settings (tg_id, should_notify, settings) VALUES ({tg_id}, 0, '{default_settings}')")

    conn.commit()
    conn.close()
```

Рис. 9 - Функция set_client

```python
def enable_notifications(user_id: int):
    db.change_notification_settings(user_id, "should_notify", "True")


def disable_notifications(user_id: int):
    db.change_notification_settings(user_id, "should_notify", "False")
```

Рис. 10 - Функции enable_notifications и disable_notifications

```python
def change_notification_settings(tg_id: int, changed_setting: str, new_value: str):
    conn = sqlite3.connect('weather.db')

    if changed_setting == "should_notify":
        if new_value == "True":
            conn.execute(f"UPDATE notify_settings SET should_notify = 1 WHERE tg_id = {tg_id}")
        else:
            conn.execute(f"UPDATE notify_settings SET should_notify = 0 WHERE tg_id = {tg_id}")
        conn.commit()
        conn.close()
```

Рис. 11 - Функция change_notification_settings

```python
def string_to_dict(s: str):
    settings = s.split()
    settings_dict = {}
    for setting in settings:
        key, value = setting.split(":")
        settings_dict[key] = value
    return settings_dict


def dict_to_string(d: dict):
    keys = list(d.keys())
    values = list(d.values())
    s = ""
    for i in range(len(d)):
        s += keys[i] + ":" + values[i] + " "
    return s
```

Рис. 12 - Функции string_to_dict и dict_to_string

```python
def send_notifications_real_true_func():
    pinged_users = []
    unpinged_users = []
    users_to_ping = db.check_notifications()
    if users_to_ping:
        for user_id in users_to_ping:
            user_settings = db.get_all_notification_settings(user_id)
            if user_settings.get(user_id)[0] == True:
                print("Sent notification to:" + str(user_id))
                pinged_users.append(user_id)
            else:
                unpinged_users.append(user_id)
        return pinged_users, unpinged_users
    else:
        return "No users to ping"
```

Рис. 13 - Функция send_notifications_real_true_func

```python
def get_all_notification_settings(tg_id: int):
    conn = sqlite3.connect('weather.db')

    user_settings = conn.execute(f"SELECT n.settings FROM notify_settings n WHERE n.tg_id = {tg_id}").fetchone()[0]
    should_notify = conn.execute(f"SELECT n.should_notify FROM notify_settings n WHERE n.tg_id = {tg_id}").fetchone()[0] == 1
    settings_dict = string_to_dict(user_settings)
    res = {tg_id: [should_notify, settings_dict]}
    return res
```

Рис. 14 - Функция get_all_notification_settings

```python
def check_notifications():
    cities = get_cities()
    users_to_ping = []
    for i in range(len(cities)):
        city = cities[i][0]
        temp = get_temp(city)
        current_temp = get_current_temp(city)
        if abs(temp - current_temp) >= notification_temp_sensitivity:
            clients = get_clients(city)
            for k in range(len(clients)):
                users_to_ping.append(clients[k][0])
        set_temp(city, current_temp)

    #refresh_weather()
    return users_to_ping
```

Рис. 15 - Функция check_notifications

```python
def get_cities():
    conn = sqlite3.connect('weather.db')
    cities = conn.execute("SELECT t.city FROM temperatures t WHERE t.city != 'None'").fetchall()
    conn.close()
    return cities


def get_temp(city: str):
    conn = sqlite3.connect('weather.db')
    temp = conn.execute(f"SELECT t.temperature FROM temperatures t WHERE t.city = '{city}'").fetchone()
    conn.close()
    return temp[0]
```

Рис. 16 - Функции get_cities и get_temp

```python
def get_current_temp(city: str):
    res = requests.get(
        f"http://api.weatherapi.com/v1/current.json?key={apikey}&q={city}"
    )
    data = res.json()
    if data.get("error"):
        return "None"

    current = data.get("current")

    return(
        float(current.get("temp_c"))
    )
```

Рис. 16 - Функция get_current_temp

После добавления нужных функций все тесты, включая новый, были запущены и успешно пройдены.

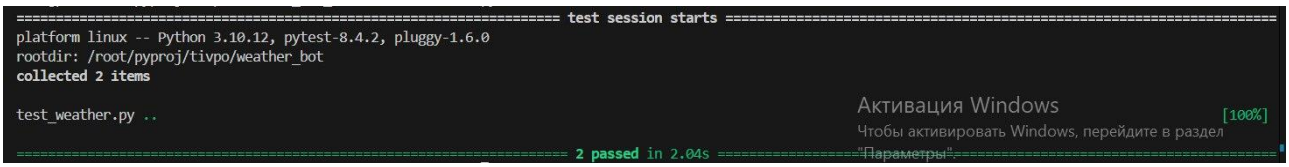


Рис. 15 – Результат теста

3.2. Этап 2: ATDD — Разработка через приёмочные тесты

После реализации основной логики фокус сместился на пользовательские сценарии. С помощью ATDD были определены и автоматизированы ключевые требования к продукту.

- **Сценарий: «При резком изменении температуры — отправить уведомление».**
  Пользователь сохраняет свой город. Затем в его городе резко меняется температура, о чем программа уведомляет пользователя. Этот сценарий проверяет работу функции send_notifications_real_true_func.

```
82
83  def test_notifications():
84      set_client(123, "Брянск")
85      enable_notifications(123)
86      set_temp("Брянск", 500)
87      pinged_users, unpinged_users = send_notifications_real_true_func()
88      assert 123 in pinged_users
```

Рис. 16 - Приёмочный тест test_notifications

Тест прошёл сразу, так как необходимая логика уже была реализована на этапе TDD

```
============================= test session starts ==============================
platform linux -- Python 3.10.12, pytest-8.4.2, pluggy-1.6.0
rootdir: /root/pyproj/tivpo/weather_bot
collected 2 items

test_weather.py ..                                                       [100%]

============================== 2 passed in 2.04s ===============================
```


Рис. 17 – Результат теста

3.3. Этап 3: BDD — Разработка через поведение

После того как ключевые пользовательские сценарии были реализованы и проверены приёмочными тестами, был применён подход BDD для создания "живой документации". Цель BDD — описать поведение системы на естественном языке, понятном как разработчикам, так и нетехническим специалистам (например, менеджерам или заказчикам).

Для этого был сформулирован сценарий отправки уведомления на языке Gherkin.

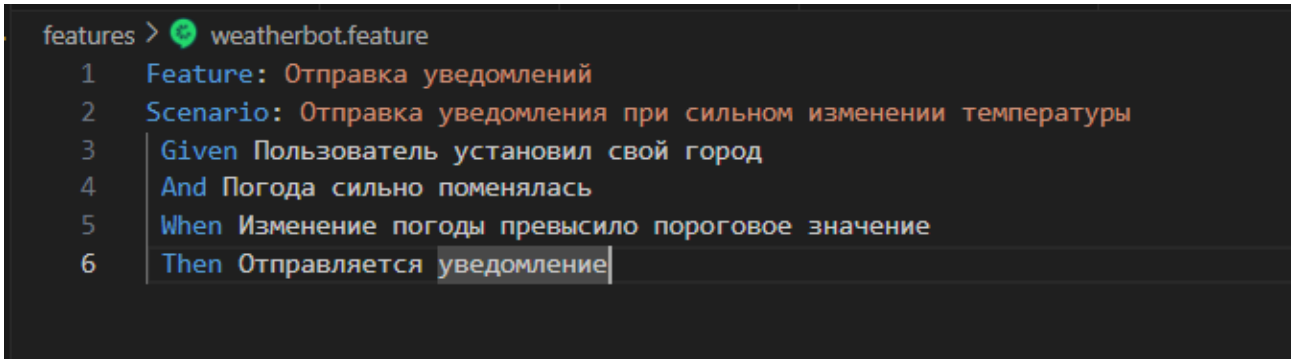


Рис. 18 – Сценарий BDD на языке Gherkin

Этот сценарий на естественном языке точно описывает поведение, которое было реализовано на предыдущем этапе (ATDD). Приёмочный тест test_notifications (Рис. 16) является прямой технической реализацией этого BDD-сценария.

Таким образом, BDD-сценарий служит мостом между бизнес-требованиями и кодом, делая систему более понятной и прозрачной.

Шаг 3.3.2: Автоматизация сценария с помощью Behave

Для того чтобы "запустить" текстовый сценарий, используется фреймворк Behave. Он связывает каждую строчку Gherkin-сценария с функцией на Python. Для этого в папке features/steps/ был создан файл weather_steps.py, содержащий "шаги" — код, реализующий логику сценария.

features > steps > weather_steps.py > step_iml_check_notifications

```python
from behave import given, when, then
from main import send_notifications_real_true_func, enable_notifications
from db import set_temp, set_client

@given('Пользователь 123 установил свой город Брянск и включил оповещения')
def step_impl_set_client(context):
    set_client(123, "Брянск")
    enable_notifications(123)

@when('Погода поменялась на {temp}')
def step_impl_change_weather(context, temp):
    set_temp("Брянск", temp)

@then('Пользователю отправляется уведомление')
def step_iml_check_notifications(context):
    pinged_users, unpinged_users = send_notifications_real_true_func()
    assert 123 in pinged_users
```

Рис. 19 – Реализация шагов в features/steps/weather_steps.py

Шаг 3.3.3: Запуск и проверка автоматизированного сценария

Для запуска теста достаточно выполнить команду behave в корневой папке проекта. Фреймворк автоматически найдет .feature файлы и соответствующие им шаги.

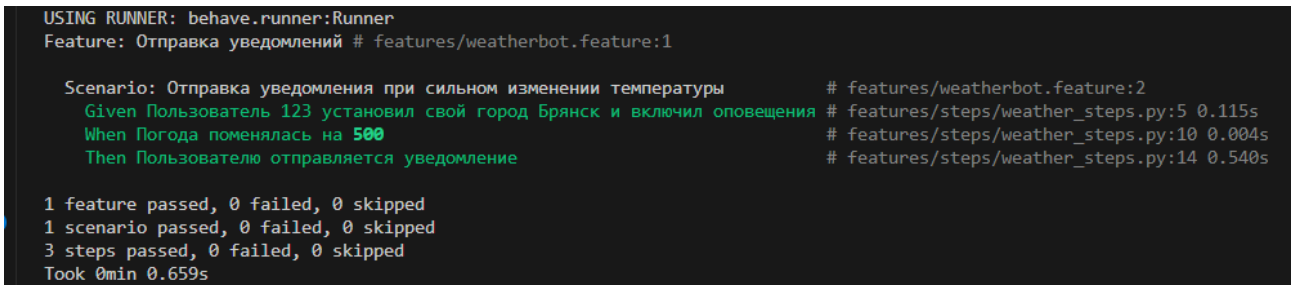


Рис. 20 – Результат успешного теста через Behave

3.4. Этап 4: SDD — Спецификация на примерах

Для дальнейшего уточнения работы функции оповещений был применён подход SDD. Он требует создания явных примеров, которые затем преобразуются в автоматизированные тесты. Это обеспечивает точное соответствие реализации ожиданиям. Для функции оповещений была составлена следующая спецификация в виде таблицы.

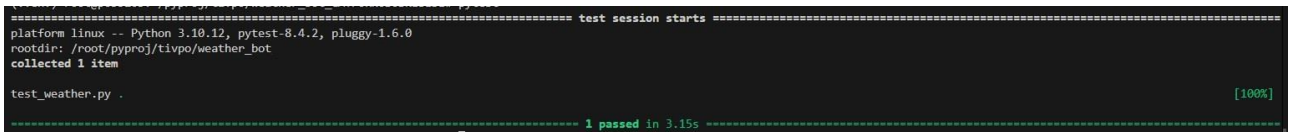
Таблица 1 — Спецификация для функции оповещений на примерах

Исходная температура	**Температура после обновления**	**Ожидаемый результат (отправляем оповещению пользователю или нет)**
500	Температура в городе	Да
-500	Температура в городе	Да
Температура в городе	Температура в городе	Нет
Температура в городе + 0.5	Температура в городе	Нет

На основе этой таблицы был создан автоматизированный тест. Этот тест напрямую, пример за примером, проверяет поведение системы.

```python
from main import send_notifications_real_true_func, enable_notifications, disable_notifications
from weather_api import get_forecast, get_current_temp
import os
from db import set_temp, set_client
#Сосиска


def test_notif_sdd():
    set_client(123, "Брянск")
    enable_notifications(123)

    set_temp("Брянск", 500)
    pinged_users, unpinged_users = send_notifications_real_true_func()
    assert 123 in pinged_users

    set_temp("Брянск", -500)
    pinged_users, unpinged_users = send_notifications_real_true_func()
    assert 123 in pinged_users

    temp = get_current_temp("Брянск")
    set_temp("Брянск", temp)
    s = send_notifications_real_true_func()
    assert s == "No users to ping"

    temp = get_current_temp("Брянск") + 0.5
    set_temp("Брянск", temp)
    s = send_notifications_real_true_func()
    assert s == "No users to ping"
```

Рис. 21 – Тест, реализующий спецификацию на примерах



```
============================= test session starts ==============================
platform linux -- Python 3.10.12, pytest-8.4.2, pluggy-1.6.0
rootdir: /root/pyproj/tivpo/weather_bot
collected 1 item

test_weather.py .                                                        [100%]

============================== 1 passed in 3.15s ===============================
```

Рис. 22 – Результат теста

Запуск этого теста подтверждает, что реализация функции send_notifications_real_true_func полностью соответствует спецификации, описанной в таблице. Таким образом, SDD позволяет создать точную, проверяемую и живую документацию, которая напрямую связана с кодом через автоматизированные тесты.

4. Результаты тестирования и анализа

4.1. **Применение методологии TDD (Test-Driven Development)**

- В рамках TDD процесс разработки строился по циклу *«красный — зеленый — рефакторинг»*.
- Сначала формулировались тесты, которые изначально завершались неуспешно.
- Затем реализовывалась минимальная логика, обеспечивающая прохождение тестов.
- После успешного выполнения тестов код подвергался рефакторингу при сохранении корректности поведения.
- Такой подход позволил выявлять ошибки на ранних этапах и гарантировать, что каждая новая функция системы сопровождалась проверкой.

4.2. **Применение методологии ATDD (Acceptance Test-Driven Development)**

- В ATDD ключевым элементом стало согласование критериев приемки с «заказчиком» до начала реализации.
- Критерии были зафиксированы в виде приемочных тестов, которые служили ориентиром для разработчиков и тестировщиков.

4.3. **Применение методологии BDD (Behavior-Driven Development)**

- BDD позволила описывать поведение системы на естественном языке с использованием формата Gherkin.
- Сценарии вида *Given-When-Then* обеспечили прозрачность требований и сделали спецификации понятными как для разработчиков, так и для пользователей.

4.4. **Применение методологии SDD (Specification by Example / Specification-Driven Development)**

- В SDD спецификации фиксировались в виде таблиц примеров, связывающих входные данные и ожидаемые результаты.
- Эти таблицы стали основой для автоматизированных тестов и позволили устранить двусмысленность требований.

4.5. **Общий анализ результатов**

- Все методологии продемонстрировали эффективность интеграции тестирования в процесс разработки.
- TDD обеспечила надежность на уровне модулей.
- ATDD позволила согласовать ожидания с точки зрения бизнеса.
- BDD сделала спецификации доступными для всех участников проекта.
- SDD устраняет двусмысленность требований за счет конкретных примеров.

5. Заключение

5.1. **Общая оценка качества программного продукта**

Программный продукт полностью реализует все заявленные функции: получение прогноза, установка оповещений, уведомление при изменениях Система успешно прошла все модульные и приёмочные тесты, которые были созданы на основе заранее определённых спецификаций и пользовательских сценариев. Это подтверждает корректность работы каждой функции и стабильность продукта в целом, включая сохранение данных между сессиями.

5.2. **Оценка качества документации**

- Документация по проекту создана в формате "живой документации" и представлена в виде BDD-сценария на естественном языке (Gherkin) и таблицы спецификаций на примерах (SDD).
- Такой подход обеспечивает полную прозрачность требований к функциональности и служит единым источником правды, облегчая коммуникацию между всеми участниками проекта.

5.3. **Выводы о проделанной работе**

- Применение комплекса методологий (TDD, ATDD, BDD, SDD) позволило выстроить процесс разработки, в котором тестирование является неотъемлемой частью, а не заключительным этапом. Это привело к созданию более надёжного и поддерживаемого кода.
- Разработанная архитектура и наличие полного набора тестов делают приложение готовым к дальнейшему развитию. Например, можно легко добавить новую функциональность, будучи уверенным, что существующая логика не будет нарушена.
- Полученный практический опыт подтверждает высокую эффективность разработки через тестирование для создания качественных программных продуктов.