|  |
| --- |
|  |
| 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

по дисциплине «Тестирование и верификация ПО»

|  |  |
| --- | --- |
| **Выполнил:**  Студент группы ИКБО-15-22 | Ератин Н.В. |
| **Проверил:**  Доцент | Чернов Е.А. |

2024 г.

**Оглавление**

[МЕТОДОЛОГИЯ TDD 3](#_Toc182747403)

[**ЭТАП №1. СОЗДАНИЕ ТЕСТА.** 3](#_Toc182747404)

[**ЭТАП №2. ЗАПУСК ТЕСТОВ.** 5](#_Toc182747405)

[**ЭТАП №3. РЕАЛИЗАЦИЯ ФУНКЦИИ.** 5](#_Toc182747406)

[**ЭТАП №4. ПОВТОРНЫЙ ЗАПУСК ТЕСТОВ.** 7](#_Toc182747407)

[МЕТОДОЛОГИЯ BDD 7](#_Toc182747408)

[**ЭТАП №1. ОПИСАНИЕ СЦЕНАРИЕВ BDD.** 8](#_Toc182747409)

[**ЭТАП №2. АВТОМАТИЗАЦИЯ СЦЕНАРИЕВ BDD.** 8](#_Toc182747410)

[**ЭТАП №3. ЗАПУСК ТЕСТОВ.** 10](#_Toc182747411)

[Заключение: 10](#_Toc182747412)

# МЕТОДОЛОГИЯ TDD

TDD, или Test-Driven Development (Разработка, управляемая тестами), - это методология разработки программного обеспечения, которая подразумевает создание тестов для функциональности ПО до того, как эта функциональность будет фактически реализована. TDD представляет собой циклический процесс, который помогает разработчикам создавать высококачественное, надежное ПО.

## **ЭТАП №1. СОЗДАНИЕ ТЕСТА.**

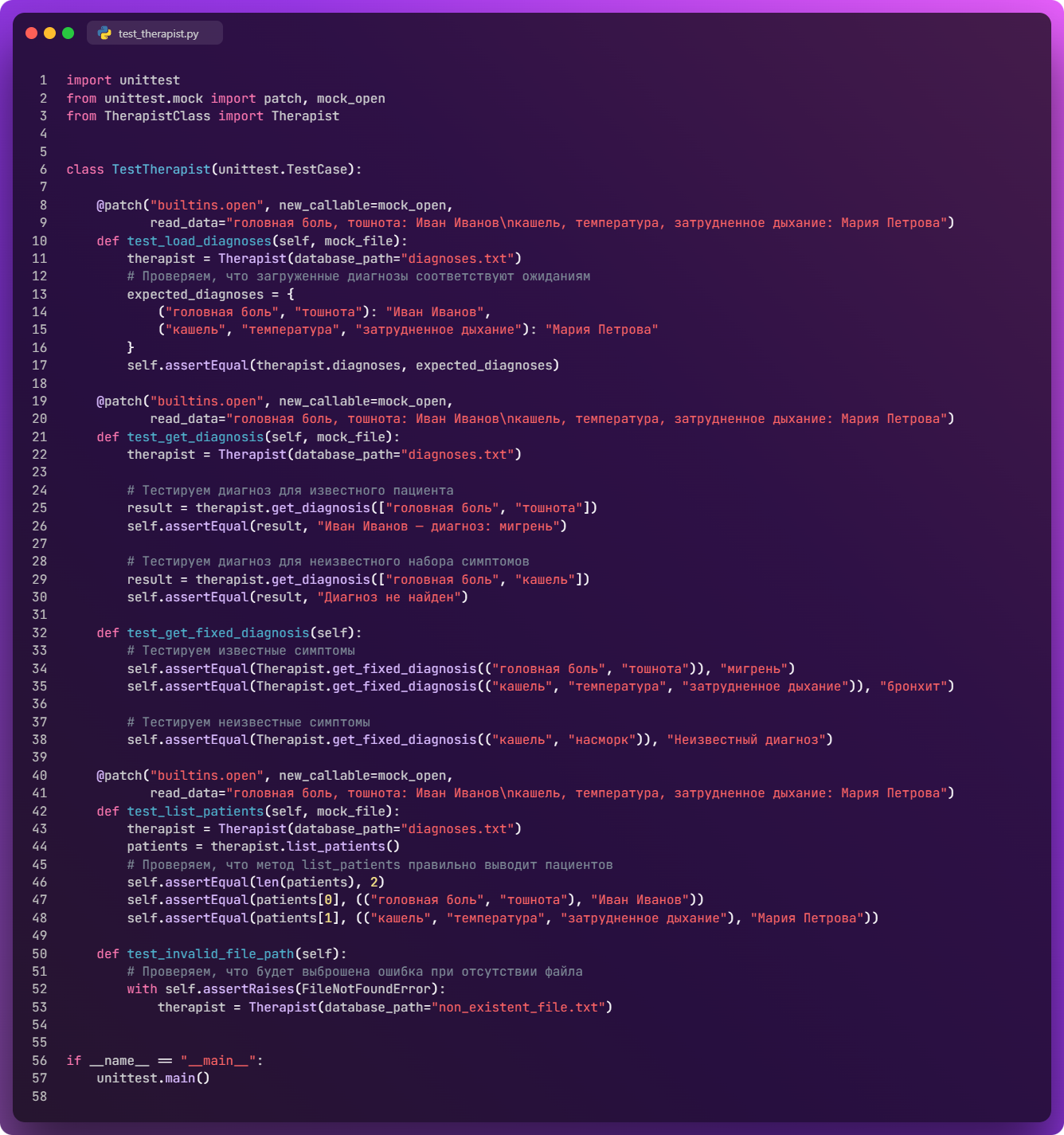


Рисунок 1 – Примеры тестов методологии TDD

*Таблица 1 – Листинг тестов методологии TDD*

|  |
| --- |
| import unittest from unittest.mock import patch, mock\_open from TherapistClass import Therapist   class TestTherapist(unittest.TestCase):   @patch("builtins.open", new\_callable=mock\_open,  read\_data="головная боль, тошнота: Иван Иванов\nкашель, температура, затрудненное дыхание: Мария Петрова")  def test\_load\_diagnoses(self, mock\_file):  therapist = Therapist(database\_path="diagnoses.txt")  # Проверяем, что загруженные диагнозы соответствуют ожиданиям  expected\_diagnoses = {  ("головная боль", "тошнота"): "Иван Иванов",  ("кашель", "температура", "затрудненное дыхание"): "Мария Петрова"  }  self.assertEqual(therapist.diagnoses, expected\_diagnoses)   @patch("builtins.open", new\_callable=mock\_open,  read\_data="головная боль, тошнота: Иван Иванов\nкашель, температура, затрудненное дыхание: Мария Петрова")  def test\_get\_diagnosis(self, mock\_file):  therapist = Therapist(database\_path="diagnoses.txt")   # Тестируем диагноз для известного пациента  result = therapist.get\_diagnosis(["головная боль", "тошнота"])  self.assertEqual(result, "Иван Иванов — диагноз: мигрень")   # Тестируем диагноз для неизвестного набора симптомов  result = therapist.get\_diagnosis(["головная боль", "кашель"])  self.assertEqual(result, "Диагноз не найден")   def test\_get\_fixed\_diagnosis(self):  # Тестируем известные симптомы  self.assertEqual(Therapist.get\_fixed\_diagnosis(("головная боль", "тошнота")), "мигрень")  self.assertEqual(Therapist.get\_fixed\_diagnosis(("кашель", "температура", "затрудненное дыхание")), "бронхит")   # Тестируем неизвестные симптомы  self.assertEqual(Therapist.get\_fixed\_diagnosis(("кашель", "насморк")), "Неизвестный диагноз")   @patch("builtins.open", new\_callable=mock\_open,  read\_data="головная боль, тошнота: Иван Иванов\nкашель, температура, затрудненное дыхание: Мария Петрова")  def test\_list\_patients(self, mock\_file):  therapist = Therapist(database\_path="diagnoses.txt")  patients = therapist.list\_patients()  # Проверяем, что метод list\_patients правильно выводит пациентов  self.assertEqual(len(patients), 2)  self.assertEqual(patients[0], (("головная боль", "тошнота"), "Иван Иванов"))  self.assertEqual(patients[1], (("кашель", "температура", "затрудненное дыхание"), "Мария Петрова"))  def test\_invalid\_file\_path(self):  # Проверяем, что будет выброшена ошибка при отсутствии файла  with self.assertRaises(FileNotFoundError):  therapist = Therapist(database\_path="non\_existent\_file.txt")   if \_\_name\_\_ == "\_\_main\_\_":  unittest.main() |

## **ЭТАП №2. ЗАПУСК ТЕСТОВ.**

На этом этапе запускаются тесты. Так как функции еще не существуют, все они провалятся.

## **ЭТАП №3. РЕАЛИЗАЦИЯ ФУНКЦИИ.**

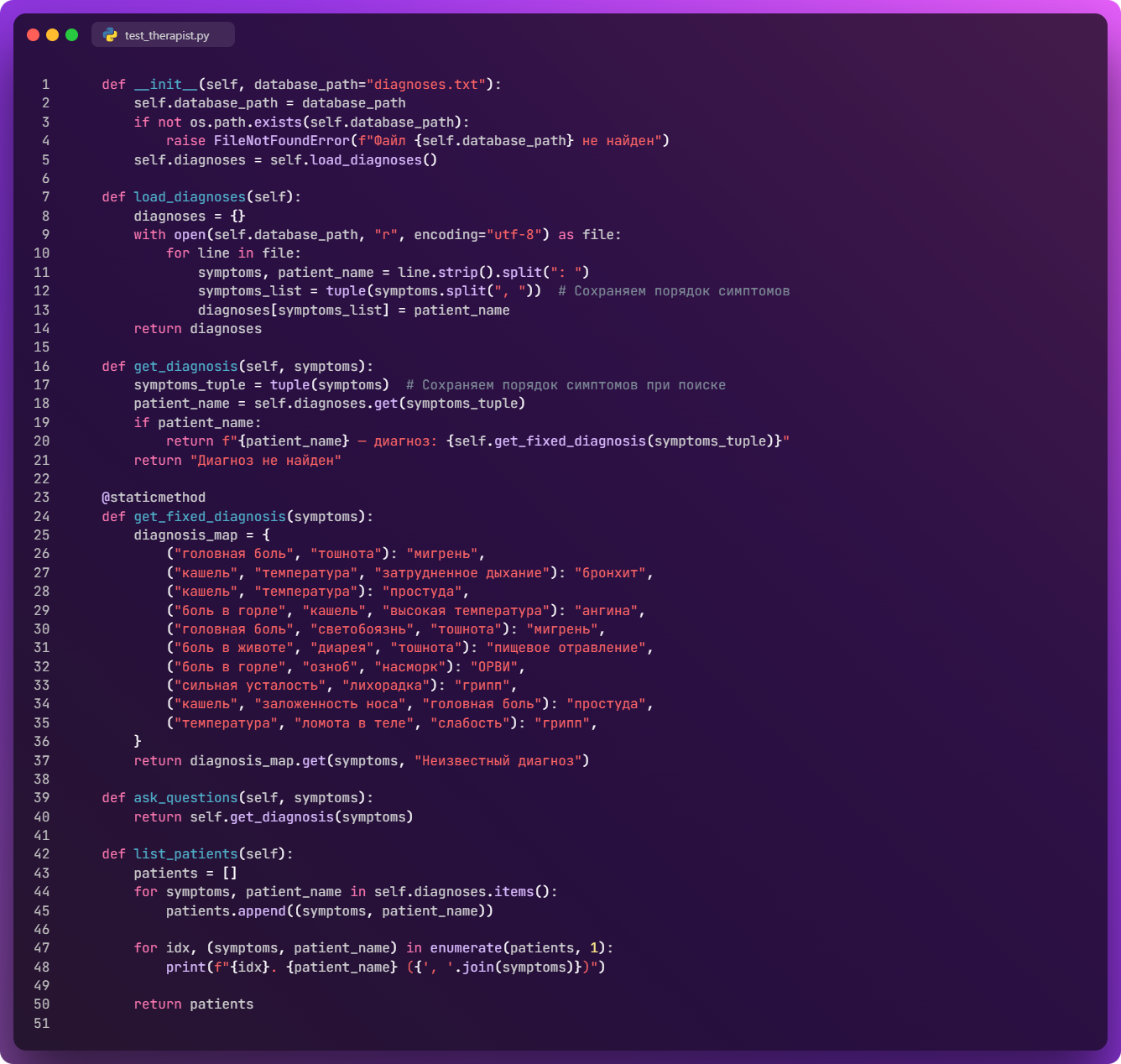


Рисунок 2 – Пример функции

*Таблица 2 – Листинг программы “Электронный терапевт”*

|  |
| --- |
| import os   class Therapist:  def \_\_init\_\_(self, database\_path="diagnoses.txt"):  self.database\_path = database\_path  if not os.path.exists(self.database\_path):  raise FileNotFoundError(f"Файл {self.database\_path} не найден")  self.diagnoses = self.load\_diagnoses()   def load\_diagnoses(self):  diagnoses = {}  with open(self.database\_path, "r", encoding="utf-8") as file:  for line in file:  symptoms, patient\_name = line.strip().split(": ")  symptoms\_list = tuple(symptoms.split(", ")) # Сохраняем порядок симптомов  diagnoses[symptoms\_list] = patient\_name  return diagnoses   def get\_diagnosis(self, symptoms):  symptoms\_tuple = tuple(symptoms) # Сохраняем порядок симптомов при поиске  patient\_name = self.diagnoses.get(symptoms\_tuple)  if patient\_name:  return f"{patient\_name} — диагноз: {self.get\_fixed\_diagnosis(symptoms\_tuple)}"  return "Диагноз не найден"   @staticmethod  def get\_fixed\_diagnosis(symptoms):  diagnosis\_map = {  ("головная боль", "тошнота"): "мигрень",  ("кашель", "температура", "затрудненное дыхание"): "бронхит",  ("кашель", "температура"): "простуда",  ("боль в горле", "кашель", "высокая температура"): "ангина",  ("головная боль", "светобоязнь", "тошнота"): "мигрень",  ("боль в животе", "диарея", "тошнота"): "пищевое отравление",  ("боль в горле", "озноб", "насморк"): "ОРВИ",  ("сильная усталость", "лихорадка"): "грипп",  ("кашель", "заложенность носа", "головная боль"): "простуда",  ("температура", "ломота в теле", "слабость"): "грипп",  }  return diagnosis\_map.get(symptoms, "Неизвестный диагноз")   def ask\_questions(self, symptoms):  return self.get\_diagnosis(symptoms)   def list\_patients(self):  patients = []  for symptoms, patient\_name in self.diagnoses.items():  patients.append((symptoms, patient\_name))   for idx, (symptoms, patient\_name) in enumerate(patients, 1):  print(f"{idx}. {patient\_name} ({', '.join(symptoms)})")   return patients   def main():  therapist = Therapist()  patients = therapist.list\_patients()   try:  patient\_number = int(input("Выберите номер пациента для получения диагноза: "))   if 1 <= patient\_number <= len(patients):  symptoms, patient\_name = patients[patient\_number - 1]  print(therapist.ask\_questions(symptoms))  else:  print("Неверный номер пациента.")  except ValueError:  print("Пожалуйста, введите корректный номер пациента.")   if \_\_name\_\_ == "\_\_main\_\_":  main() |

## **ЭТАП №4. ПОВТОРНЫЙ ЗАПУСК ТЕСТОВ.**

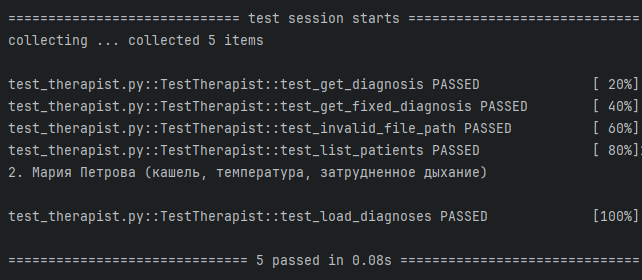


Рисунок 3 – Успешное выполнение тестов

# МЕТОДОЛОГИЯ BDD

**BDD, или Behavior Driven Development** (Разработка, ориентированная на поведение), - это методология разработки программного обеспечения, которая сосредотачивается на описании поведения программы с точки зрения её пользователей и интересующих сторон. BDD представляет собой эволюцию техники TDD (Test-Driven Development), в которой акцент делается на спецификациях поведения и участии бизнес-аналитиков и представителей заказчика в процессе разработки.

## **ЭТАП №1. ОПИСАНИЕ СЦЕНАРИЕВ BDD.**

На первом этапе создаётся описание сценариев BDD для функциональности программы.

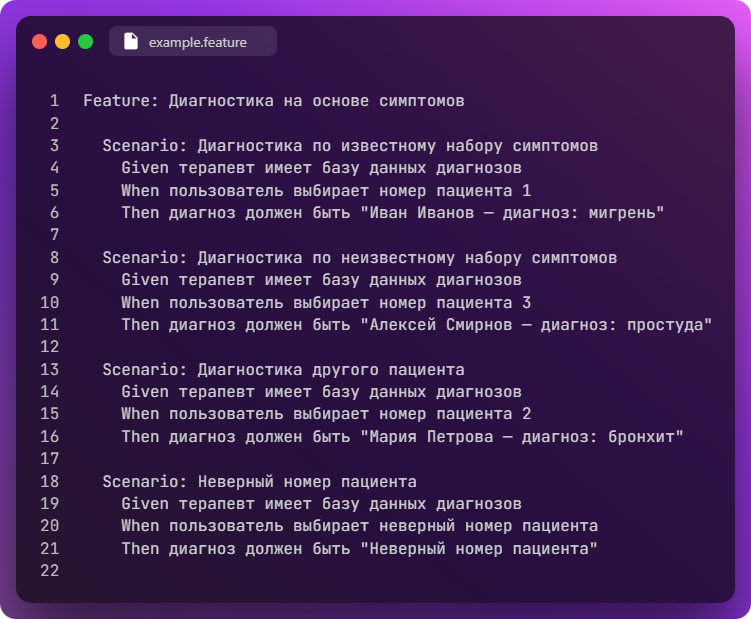


Рисунок 4 – Описание сценариев

## **ЭТАП №2. АВТОМАТИЗАЦИЯ СЦЕНАРИЕВ BDD.**

На следующем шаге создаются автоматизированные тесты для каждого сценария, используя фреймворк для тестирования, который поддерживает BDD, в нашем случае это Behave для Python.

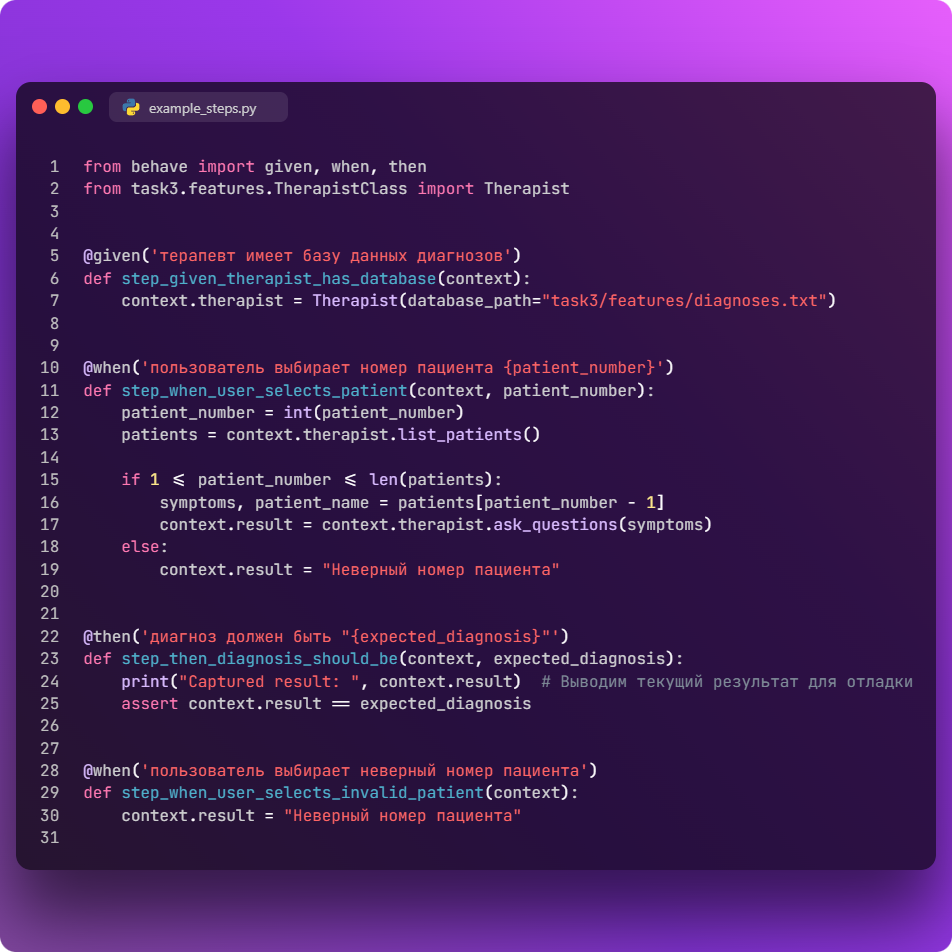


Рисунок 5 – Описание тестов

*Таблица 3 – Листинг тестов методологии BDD*

|  |
| --- |
| from behave import given, when, then from task3.features.TherapistClass import Therapist   @given('терапевт имеет базу данных диагнозов') def step\_given\_therapist\_has\_database(context):  context.therapist = Therapist(database\_path="task3/features/diagnoses.txt")   @when('пользователь выбирает номер пациента {patient\_number}') def step\_when\_user\_selects\_patient(context, patient\_number):  patient\_number = int(patient\_number)  patients = context.therapist.list\_patients()   if 1 <= patient\_number <= len(patients):  symptoms, patient\_name = patients[patient\_number - 1]  context.result = context.therapist.ask\_questions(symptoms)  else:  context.result = "Неверный номер пациента"   @then('диагноз должен быть "{expected\_diagnosis}"') def step\_then\_diagnosis\_should\_be(context, expected\_diagnosis):  print("Captured result: ", context.result) # Выводим текущий результат для отладки  assert context.result == expected\_diagnosis   @when('пользователь выбирает неверный номер пациента') def step\_when\_user\_selects\_invalid\_patient(context):  context.result = "Неверный номер пациента"  ] |

## **ЭТАП №3. ЗАПУСК ТЕСТОВ.**

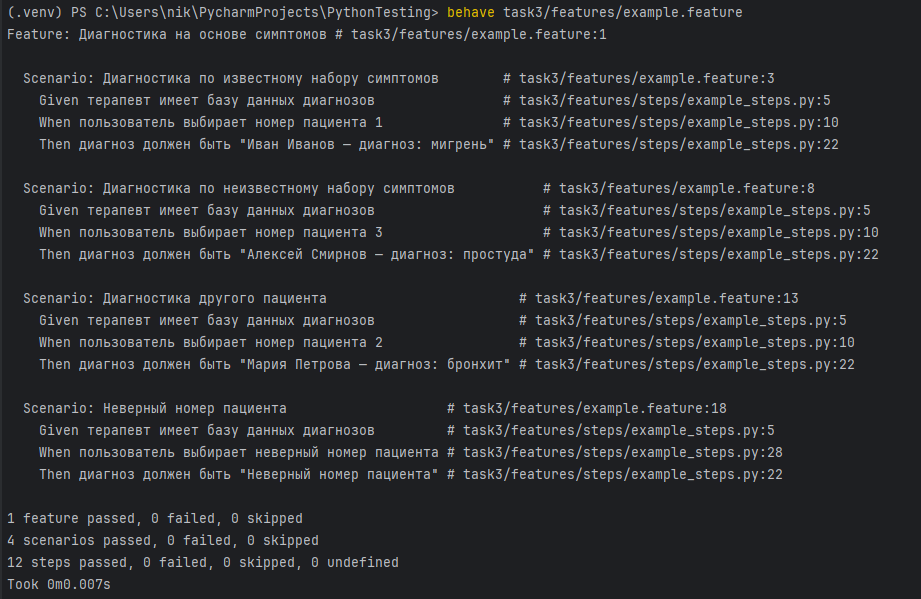


Рисунок 6 – Успешное выполнение тестов

# Заключение:

В ходе работы были получены практический опыт применения методологий TDD (Test-Driven Development) и BDD (Behavior-Driven Development) в процессе разработки программного обеспечения.