Петрозаводский государственный университет
Институт математики и информационных технологий
Кафедра прикладной математики и кибернетики

Направление подготовки бакалавриата

09.03.04 – Программная инженерия

Профиль направления подготовки бакалавриата

Системное и прикладное программное обеспечение

Отчет по дисциплине «Верификация ПО»

Разработка программного обеспечения для прототипа кассы самообслуживания

Выполнила:
студентка 4 курса группы 22407

Е. Ф. Волкова _____
подпись

Руководитель:
К. А. Кулаков

_____
подпись

Итоговая оценка:

_____
оценка

Петрозаводск — 2023

Содержание

1 Объект тестирования

1.1 Описание приложения

Прототип кассы самообслуживания - это система, которая позволяет покупателям сканировать и оплачивать выбранные товары без помощи кассира.

Система использует несколько аппаратных компонентов, такие как RFID-считыватели, метки, а также консольное приложение для взаимодействия пользователя с данной системой.

В рамках ВКР планируется разработать программное обеспечение (ПО) для прототипа кассы самообслуживания, предназначенного для автоматизации процесса совершения покупок в магазине.

Принцип работы прототипа:

1. Покупатель выбирает товары, которые он хочет приобрести.
2. Покупатель подносит товар к считывателю.
3. Система сканирует каждый товар и проверяет его на наличие в базе данных.
4. Система отображает список товаров на экране.
5. Покупатель может ознакомиться со списком товаров и подтвердить покупку.
6. В случае оплаты, выводится сообщение об успешной покупке.
7. После покупки товары удаляются из базы данных.

Таким образом, ПО имеет следующие основные функции:

1. Сканирование RFID-меток считывателем.
2. Вывод на экран списка сканированных товаров (меток).
3. Хранение товаров (меток) с помощью базы данных.
4. Возможность покупки - удаление меток из базы данных.

1.2 Архитектура прототипа

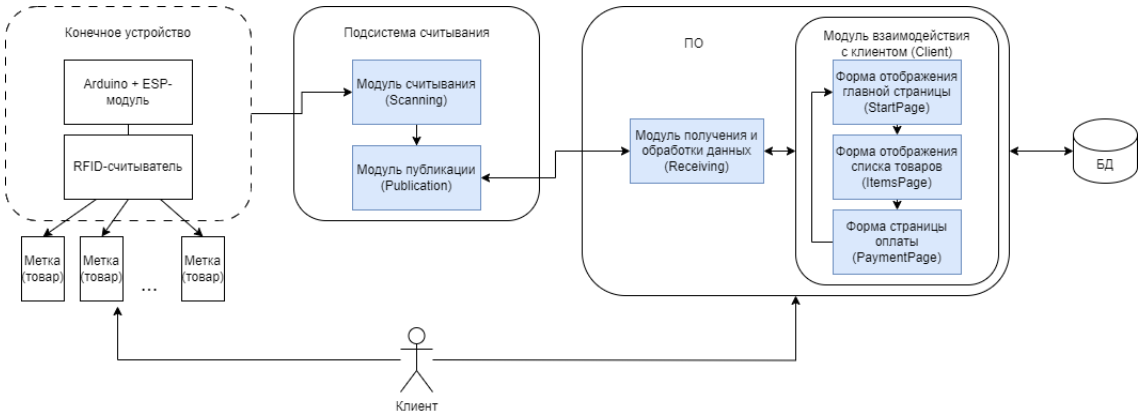

Рис. 1 – Архитектура системы

Тестируемые модули: модуль считывания, модуль публикации, модуль получения и обработки данных, а также модуль взаимодействия с клиентом.

1.3 Описание модулей и методов

В качестве **конечного устройства** используется RFID-считыватель (модуль RC522), получающий информацию о RFID-метках, в качестве контроллера – плата Arduino, также для подключения к интернету – модуль ESP-01 на чипе ESP8266.

Подсистема считывания:

1. **Модуль считывания** (Scanning) отвечает за сканирование RFID-меток и получение данных о них (UID - уникальный идентификатор) с RFID-считывателя.

    - void setup() - функция для инициализации модулей, подключения и настройки MQTT сервера. Косвенные входные параметры: ssid[], pass[], server[] - SSID и пароль WiFi сети, адрес и порт сервера, выходные - состояние MQTT клиента и его статус подключения.
    - void loop() - функция для проверки соединения, сканирования меток и соответствия типу метки. Косвенные входные параметры: RFID метки, выходные параметры: UID метки.

2. **Модуль публикации** (Publication) отвечает за публикацию информации о метках.

    - void readRFID() - функция необходима для публикации сканированных меток по MQTT протоколу на платформу. Входные косвенные параметры: UID метки,

выходные параметры: сообщение о их типе, сообщение в тему о сведениях метки (если метка типа Mifare).

Связь между модулями публикации и получения данных реализуется через MQTT протокол, работающему по принципу издатель-подписчик.

ПО:

1. **Модуль получения и обработки данных** (Receiving) отвечает за принятие сообщений по MQTT протоколу, проверку на наличие товаров в базе данных и отправку данных об этих метках в модуль взаимодействия с клиентом.

   - connect_mqtt() - функция для подключения к брокеру MQTT, косвенные входные параметры: client_id - идентификатор клиента, broker - адрес брокера, port - порт, возвращает экземпляр класса mqtt_client.
   - subscribe(client: mqtt_client) - функция для подписки на тему MQTT. Входными параметрами являются client типа mqtt_client (косвенными - тема), выходными - on_message().
   - on_message(client, userdata, msg) - функция для получения сообщений и проверку на наличие товаров в БД. Входные параметры: client - это экземпляр MQTT клиента, userdata - это данные, которые были переданы при подписке на тему, msg - это объект сообщения, который передает в качестве сообщения UID метки. Функция возвращает обновленный список покупок buylist.
   - run() - функция для запуска MQTT-клиента. Косвенные входные параметры: client - это экземпляр MQTT клиента, выходные параметры: mutex (используется для синхронизации доступа к теме buylist).

2. **Модуль взаимодействия с клиентом** (Client) - модуль, отвечающий за отображение интерфейса приложения, вывод на экран списков товаров и осуществление покупки.

   - class MainApplication() - главное окно приложения, содержит контейнер для фреймов и словарь для хранения ссылок на все фреймы.

     Функции: __init__(self) инициализирует главное окно приложения и создает контейнер для фреймов. Входные параметры: объект класса, выходные параметры: экземпляр класса MainApplication.

show_frame(cont) переключает видимость фрейма на указанный. Входные параметры: cont - класс фрейма, который нужно отобразить.

- class StartPage() - класс представляет собой первую страницу приложения.

  Функции: __init__(self, parent, controller) инициализирует начальную страницу приложения. Входные параметры: parent - родительский фрейм, controller - экземпляр класса MainApplication, выходные параметры: экземпляр класса StartPage.

- class ItemsPage() - класс представляет собой страницу приложения с отображением списка покупок.

  Функции: __init__(self, parent, controller) инициализирует страницу для вывода списка товаров. Входные параметры: parent - родительский фрейм, controller - экземпляр класса MainApplication, выходные параметры: экземпляр класса ItemsPage.

  update_list(text) - функция для отображения обновленного списка покупок. Входные параметры: text - UID метки, text_field - ссылка на текстовое поле (косвенный параметр), выходные параметры: обновленное поле text_field.

- class PaymentPage() - класс представляет собой страницу оплаты.

  Функции: __init__(self, parent, controller) инициализирует страницу оплаты. Входные параметры: parent - родительский фрейм, controller - экземпляр класса MainApplication, выходные параметры: экземпляр класса PaymentPage.

  clear_list() - функция для удаления товаров из БД. Входные параметры: buylist - список товаров, выходные параметры: сообщение о статусе оплаты.

База данных (Redis) хранит все данные о RFID-метках.

2 Стратегия тестирования

2.1 Стратегия блочного тестирования

Unit-тестирование (модульное тестирование) — процесс, позволяющий проверить на корректность единицы исходного кода, наборы из одного или более программных модулей вместе с соответствующими управляющими данными, процедурами использования и обработки. Идея состоит в том, чтобы писать тесты для каждой нетривиальной функции или метода. Это позволяет достаточно быстро проверить, не привело ли очередное изменение кода к регрессии, то есть к появлению ошибок в уже оттестированных местах программы, а также облегчает обнаружение и устранение таких ошибок.

Для выполнения блочного тестирования будут использоваться библиотека PyTest.

2.2 Стратегия интеграционного тестирования

Интеграционное тестирование — одна из фаз тестирования программного обеспечения, при которой отдельные программные модули объединяются и тестируются в группе. Обычно интеграционное тестирование проводится после модульного тестирования и предшествует системному тестированию. Используется библиотека PyTest.

Целью интеграционного тестирования является проверка соответствия проектируемых единиц функциональным, приёмным и требованиям надежности. Тестирование этих проектируемых единиц — объединения, множества или группы модулей — выполняется через их интерфейс, с использованием тестирования «чёрного ящика».

Интеграционному тестированию подлежат следующие взаимодействия модулей:

- **Связь 1.** Модуль считывания –> Модуль публикации (считывание UID меток и передача их в модуль для публикации в тему).
- **Связь 2.** Модуль публикации –> Модуль получения и обработки данных (получение сообщения от модуля публикации и передача сообщения с данными товара в модуль получения и обработки).
- **Связь 3.** Модуль получения и обработки данных –> Модуль взаимодействия с клиентом –> База данных (получение сообщения с UID метки и передача модулю взаимодействия данных этого товара для отображения в списке покупок, если товар хранится в БД).

- **Связь 4.** Модуль взаимодействия с клиентом –> База данных (сканированный список товаров от пользователя передается в базу данных для их удаления).

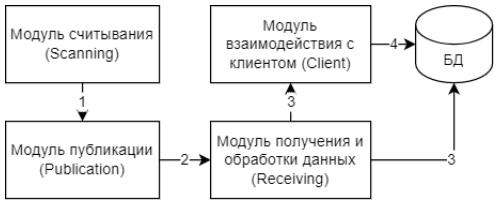

Рис. 2 – Стратегия интеграционного тестирования

2.3 Стратегия аттестационного тестирования

Аттестационное тестирование - процесс тестирования пользователем, являющийся по сути первичным обнаружением ошибок. Методы ручного тестирования достаточно эффективны с точки зрения нахождения ошибок, так что один или несколько из них должны использоваться в каждом программном проекте.

Аттестационное тестирование будет проводиться вручную.

2.4 Стратегия нагрузочного тестирования

Нагрузочное тестирование – это процесс умышленной нагрузки системы, с целью определения показателей производительности, времени отклика, проверки соответствия требованиям, которые были предъявлены к данной системе.

Целью данного тестирования является оценка производительности и работоспособности тестируемого модуля. Для такого вида тестирования будет использоваться Locust. Locust - это открытый инструмент для нагрузочного тестирования, который позволяет написать сценарии тестирования на Python. Он позволяет определить поведение виртуальных пользователей в коде и генерировать нагрузку на вашу систему.

Данные, на которых будет проводиться нагрузочный тест:

- Список из 5 RFID-меток
- Список из 10 RFID-меток
- Список из 50 RFID-меток
- Список из 100 RFID-меток

Каждая метка отправляется в модуль публикации с таймаутом равным в 1 секунду.

3 Блочное тестирование

Тест Б1 [позитивный]

Объект тестирования	Модуль получения и обработки данных, метод connect_mqtt()
Описание	Проверяет, успешно ли клиентская часть подключилась к брокеру MQTT
Начальное состояние	Брокер запущен
Входные данные	Адрес брокера broker = '192.168.31.226', порт port = 1883, client_id = 1
Ожидаемый результат	Создает и возвращает экземпляр класса Client

Тест Б2 [негативный]

Объект тестирования	Модуль получения и обработки данных, метод connect_mqtt()
Описание	Проверяет, что невозможно подключиться к брокеру при вводе неправильных данных брокера (адреса)
Начальное состояние	Брокер запущен
Входные данные	Адрес брокера broker = '1.1.1.1', порт port = 1883, client_id = 1
Ожидаемый результат	Вызов исключения ConnectionError, вывод сообщения «Ошибка подключения»

Тест Б3 [негативный]

Объект тестирования	Модуль получения и обработки данных, метод connect_mqtt()
Описание	Проверяет, что невозможно подключиться к брокеру при вводе неправильных данных брокера (порта)
Начальное состояние	Брокер запущен
Входные данные	Адрес брокера broker = '192.168.31.226', порт port = 188843, client_id = 1
Ожидаемый результат	Вызов исключения ConnectionError, вывод сообщения «Ошибка подключения»

Тест Б4 [позитивный]

Объект тестирования	Модуль получения и обработки данных, метод subscribe()
Описание	Проверяет, успешно ли функция подписывается на тему MQTT
Начальное состояние	Брокер запущен
Входные данные	Экземпляр класса Client, название темы topic = "/buylist"
Ожидаемый результат	Функция устанавливает обработчик сообщений и подписывается на тему

Тест Б5 [позитивный]

Объект тестирования	Модуль публикации, метод readRFID()
Описание	Проверяет, успешно ли функция readRFID считывает RFID-метку и публикует UID
Начальное состояние	Брокер запущен, подписка на тему buylist, наличие товара в БД с UID = ’35 7 185 22’
Входные данные	RFID-метка типа Mifare с UID = ’35 7 185 22’
Ожидаемый результат	Функция считывает RFID-метку и публикует UID в консоль (’35 7 185 22’)

Тест Б6 [негативный]

Объект тестирования	Модуль публикации, метод readRFID()
Описание	Проверяет, что невозможно сканировать метку другого типа
Начальное состояние	Брокер запущен, наличие товаров в БД, подписка на тему buylist
Входные данные	RFID-метка типа не Mifare (например, Ucode)
Ожидаемый результат	Вывод сообщения в консоли "Your tag is not of type MIFARE Classic."

Тест Б7 [позитивный]

Объект тестирования	Модуль взаимодействия с клиентом, метод clear_list()
Описание	Проверяет, успешно ли функция очищает список покупок и выводит сообщение об успешной покупке
Начальное состояние	Брокер запущен, подписка на тему buylist, наличие товаров в БД
Входные данные	Список покупок buylist
Ожидаемый результат	Функция очищает список покупок, len(buylist) = 0

Тест Б8 [позитивный]

Объект тестирования	Модуль взаимодействия с клиентом, метод update_list()
Описание	Проверяет, успешно ли функция обновляет список покупок при получении нового сообщения от MQTT брокера
Начальное состояние	Брокер запущен, наличие товара в БД с UID = '35 7 185 22', подписка на тему buylist, сканирование метки типа Mifare
Входные данные	UID метки = '35 7 185 22'
Ожидаемый результат	Обновляется список покупок buylist, отображаются данные сканированной метки: название 'Ручка Parker', цена '35'

4 Интеграционное тестирование

Тест И1 [позитивный]

Объект тестирования	Связь 1 : модуль считывания, модуль публикации
Описание	Проверка, что модуль считывания сканирует RFID-метки и передает их в модуль для публикации
Начальное состояние	Брокер запущен
Входные данные	Метка типа Mifare с UID = ’35 7 185 22’
Ожидаемый результат	В функцию readRFID() от функции loop() передается UID метки = ’35 7 185 22’, UID метки выводится в консоль (’35 7 185 22’)

Тест И2 [позитивный]

Объект тестирования	Связь 2 : модуль публикации, модуль получения и обработки данных
Описание	Проверка, что модуль публикации публикует UID меток типа Mifare в тему buylist, а модуль получения и обработки получает сообщение с темы о сведениях этой метки
Начальное состояние	Брокер запущен
Входные данные	Метка типа Mifare с UID = ’35 7 185 22’
Ожидаемый результат	Функция readRFID() передает UID метки = ’35 7 185 22’ в тему buylist, on_message() получает сообщение с данными метки

Тест И3 [позитивный]

Объект тестирования	Связь 3 : модуль получения и обработки данных, модуль взаимодействия с клиентом, база данных
Описание	Проверка, что on_message() передает в функцию update_list данные метки, если метка присутствует в базе данных
Начальное состояние	Брокер запущен, наличие товаров в БД (например, товара с UID = ’35 7 185 22’), подписка на тему buylist
Входные данные	UID метки, хранящийся в БД (UID = ’35 7 185 22’)
Ожидаемый результат	on_message() передает UID = ’35 7 185 22’, а update_list() считывает данные этой метки: name = ’Ручка Parker’, price = ’35’

Тест И4 [позитивный]

Объект тестирования	Связь 4 : модуль взаимодействия с клиентом, база данных
Описание	Проверить, что после очищения списка с помощью clear_list() данные меток удаляются из базы данных
Начальное состояние	Брокер запущен, наличие товаров в БД
Входные данные	Список товаров buylist
Ожидаемый результат	После нажатия на кнопку «Оплатить», данные метки удаляются из БД. В консоль выводится сообщение о удаливших метках.

5 Аттестационное тестирование

Тест А1 [позитивный]

Объект тестирования	Функция для отображения главной страницы
Описание	Проверить, что при запуске приложения открывается главная страница
Начальное состояние	Брокер запущен
Процесс	Пользователь запускает приложение
Ожидаемый результат	Открывается главная страница

Тест А2 [позитивный]

Объект тестирования	Функции сканирования и отображения списка товаров
Описание	Проверить, что при сканировании товара с RFID-меткой типа Mifare товар отображается на экране
Начальное состояние	Брокер запущен, наличие товара в БД (например, с UID = '35 7 185 22'), подписка на тему buylist
Процесс	Пользователь подносит товар к считывателю. Например, товар с UID = '35 7 185 22'
Ожидаемый результат	Данные товара отображаются на экране. Например, название 'Ручка Parker', цена '35'

Тест А3 [негативный]

Объект тестирования	Функции сканирования и отображения списка товаров
Описание	Проверить, что при сканировании товара с RFID-меткой НЕ типа Mifare товар не отображается на экране
Начальное состояние	Брокер запущен, подписка на тему buylist
Процесс	Пользователь подносит товар к считывателю с меткой НЕ типа Mifare (например, Ucode)
Ожидаемый результат	Данные товара не отображаются на экране

Тест А4 [позитивный]

Объект тестирования	Функция осуществления покупки
Описание	Проверить, что при нажатии на кнопку «Перейти к оплате» очищается список товаров на экране
Начальное состояние	Брокер запущен, наличие товаров в БД, подписка на тему buylist
Процесс	Пользователь находится на странице отображения списков товаров, в списке покупок присутствуют сканированные товары. Пользователь нажимает на кнопку «Перейти к оплате»
Ожидаемый результат	Отображается страница оплаты, список товаров очищается

Тест А5 [позитивный]

Объект тестирования	Функция осуществления покупки
Описание	Проверить, что при нажатии на кнопку «Оплатить» очищается список товаров (удаляются данные из БД)
Начальное состояние	Брокер запущен, наличие товаров в БД, подписка на тему buylist
Процесс	Пользователь находится на странице оплаты, нажимает на кнопку «Оплатить»
Ожидаемый результат	Очищение списка товаров и отображение сообщения «Оплата прошла успешно. Спасибо за покупку!»

Тест А6 [позитивный]

Объект тестирования	Функция отображения списка товаров
Описание	Проверить, что при нажатии на кнопку «Перейти к покупкам» отображается страница для вывода списка покупок
Начальное состояние	Брокер запущен, подписка на тему buylist
Процесс	Пользователь находится на главной странице и нажимает на кнопку «Перейти к покупкам» для отображения страницы со списком сканированных товаров
Ожидаемый результат	Открывается страница для отображения списка товаров

Тест А7 [позитивный]

Объект тестирования	Функция для отображения главной страницы
Описание	Проверить, что при нажатии на кнопку «Перейти на главную страницу» отображается главная (начальная) страница приложения
Начальное состояние	Брокер запущен, подписка на тему buylist
Процесс	Пользователь находится на странице оплаты и нажимает на кнопку «Перейти на главную страницу»
Ожидаемый результат	Открывается начальная страница

6 Нагрузочное тестирование

Тест Н1

Описание	Проверить, что время вывода UID-метки в консоль не более 25мс при сканировании списка из 5 меток с заданным таймаутом в 1 сек
Начальное состояние	Брокер запущен, подписка на тему buylist, наличие всех 5 товаров в БД
Входные данные	Сгенерированный список пяти UID-меток
Ожидаемый результат	В консоль данные метки выводятся не более, чем за 25 мс

Тест Н2

Описание	Проверить, что время вывода UID-метки в консоль не более 25мс при сканировании списка из 10 меток с заданным таймаутом в 1 сек
Начальное состояние	Брокер запущен, подписка на тему buylist, наличие всех 10 товаров в БД
Входные данные	Сгенерированный список десяти UID-меток
Ожидаемый результат	В консоль данные метки выводятся не более, чем за 25 мс

Тест Н3

Описание	Проверить, что время вывода UID-метки в консоль не более 25мс при сканировании списка из 50 меток с заданным таймаутом в 1 сек
Начальное состояние	Брокер запущен, подписка на тему buylist, наличие всех 50 товаров в БД
Входные данные	Сгенерированный список пятидесяти UID-меток
Ожидаемый результат	В консоль данные метки выводятся не более, чем за 25 мс

Тест Н4

Описание	Проверить, что время вывода UID-метки в консоль не более 25мс при сканировании списка из 100 меток с заданным таймаутом в 1 сек
Начальное состояние	Брокер запущен, подписка на тему buylist, наличие всех 100 товаров в БД
Входные данные	Сгенерированный список ста UID-меток
Ожидаемый результат	В консоль данные метки выводятся не более, чем за 25 мс

7 Журнал тестирования

7.1 Блочное тестирование

Тест	Дата	Тестировщик	Попытки	Результат	Отчет об ошибке
Б1	07.12.2023	Волкова Е. Ф.	1	Пройден	
Б2	07.12.2023	Волкова Е. Ф.	2	Пройден	Отчет №1
Б3	07.12.2023	Волкова Е. Ф.	2	Пройден	Отчет №2
Б4	07.12.2023	Волкова Е. Ф.	1	Пройден	
Б5	07.12.2023	Волкова Е. Ф.	1	Пройден	
Б6	07.12.2023	Волкова Е. Ф.	1	Пройден	
Б7	07.12.2023	Волкова Е. Ф.	1	Пройден	
Б8	07.12.2023	Волкова Е. Ф.	1	Пройден	

7.2 Интеграционное тестирование

Тест	Дата	Тестировщик	Попытки	Результат	Отчет об ошибке
И1	07.12.2023	Волкова Е. Ф.	1	Пройден	
И2	07.12.2023	Волкова Е. Ф.	1	Пройден	
И3	07.12.2023	Волкова Е. Ф.	2	Пройден	Отчет №3
И4	07.12.2023	Волкова Е. Ф.	1	Пройден	

7.3 Аттестационное тестирование

Тест	Дата	Тестировщик	Попытки	Результат	Отчет об ошибке
A1	07.12.2023	Волкова Е. Ф.	1	Пройден	
A2	07.12.2023	Волкова Е. Ф.	1	Пройден	
A3	07.12.2023	Волкова Е. Ф.	1	Пройден	
A4	07.12.2023	Волкова Е. Ф.	1	Пройден	
A5	07.12.2023	Волкова Е. Ф.	1	Пройден	
A6	07.12.2023	Волкова Е. Ф.	1	Пройден	
A7	07.12.2023	Волкова Е. Ф.	1	Пройден	

7.4 Нагрузочное тестирование

Тест	Дата	Тестировщик	Попытки	Результат	Отчет об ошибке
H1	10.12.2023	Волкова Е. Ф.	1	Пройден	
H2	10.12.2023	Волкова Е. Ф.	1	Пройден	
H3	10.12.2023	Волкова Е. Ф.	1	Пройден	
H4	10.12.2023	Волкова Е. Ф.	1	Пройден	

8 Журнал ошибок

№	Дата	Тест	Ожидаемый результат	Фактический результат	Решение
1	07.12.2023	Б2	Вызов исключения ConnectionError	Ошибка подключения не обрабатывается	Добавлено исключение "except socket.error"с выводом информации об ошибке подключения
2	07.12.2023	Б3	Вызов исключения ConnectionError	Ошибка подключения не обрабатывается	Добавлено исключение "except socket.error"с выводом информации об ошибке подключения
3	07.12.2023	И3	on_message() передает UID метки, а update_list() считывает данные этой метки (price, name)	UID метки не хранится в БД	Добавлена UID метки в БД вместе с ее данными (price, name)

9 Примеры тестов

```
# Пример блочного теста
def test_clear_list(self):
    # Добавляем некоторые элементы в buylist
    buylist.append('item1')
    buylist.append('item2')
    buylist.append('item3')
    # Вызываем функцию clear_list
    clear_list()
    # Проверяем, что buylist очищен
    self.assertEqual(len(buylist), 0)
```

```
# Пример блочного теста
def test_connect_mqtt(self):
   # Создаем экземпляр mqtt_client.Client для имитации брокера
   broker = mqtt_client.Client("test_broker")
   port = 1883 # стандартный порт для MQTT
   client_id = "test_client"
   # Вызываем функцию, которую мы тестируем
   result = connect_mqtt(broker, port, client_id)
   # Проверяем, что результат является экземпляром mqtt_client.Client
   self.assertIsInstance(result, mqtt_client.Client)
   # Проверяем, что экземпляр client был подключен к брокеру
   self.assertTrue(result.is_connected())
```

```
# Пример интеграционного теста
def test_clear():
   mock_redis = mock.MagicMock()
   mock_client = mock.MagicMock()
   mock_buylist = ['uid']
   # Замена реальных объектов на моки
   with mock.patch('module.r', mock_redis), \
        mock.patch('module.buylist', mock_buylist), \
        mock.patch('module.mutex', 0):
       # Вызов функции clear_list
       clear_list()
       # Проверка, что метод delete был вызван для каждого элемента в buylist
       for uid in mock_buylist:
```

```
14              # Проверка, что из БД удалились uid
15              mock_redis.delete.assert_any_call(uid)
16          # Проверка, что список buylist был очищен
17          assert not mock_buylist
```

10 Оценка покрытия кода тестами

Для оценки покрытия кода был использован PyTest-Cov. Это плагин для фреймворка тестирования PyTest, который позволяет анализировать покрытие кода тестами. Он помогает определить, какая часть вашего кода была выполнена во время тестирования, и выявить участки кода, которые не были протестированы. Общее покрытие кода тестами составило 81% по оценкам тестирования с помощью плагина (без аттестационных тестов).

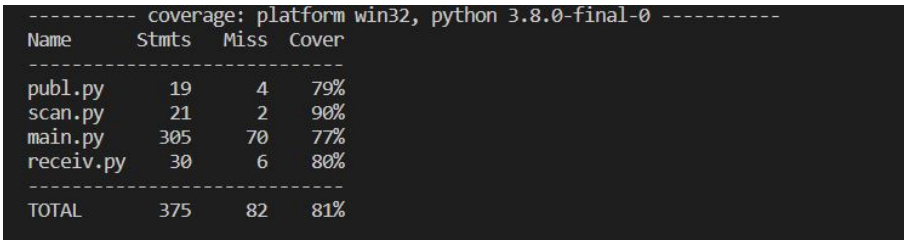

Рис. 3 – Оценка покрытия

Заключение

В ходе тестирования программного обеспечения были проведены 8 блочных, 4 интеграционных, 7 аттестационных и 4 нагрузочных теста. Тесты были направлены на проверку корректности работы ключевых функций программы. Основной целью тестирования было обеспечение надежности и эффективности работы приложения в условиях реального использования.

В ходе тестирования были выявлены 3 ошибки. Найденные ошибки существенно влияют на работу системы, но были исправлены. После исправления найденных ошибок программа является работоспособной.