



МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования

"МИРЭА - Российский технологический университет"

РТУ МИРЭА

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

ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ №2

по дисциплине

«Тестирование и верификация программного обеспечения»

Выполнили студенты группы ИКБО-04-22

Егоров Л.А.
Корольков А.Д.
Кликушин В.И.
Преснякова А.В.

Приняла ассистент

Петрова А.А.

Москва 2024

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	3
Постановка задачи.....	4
1 СОЗДАНИЕ МОДУЛЯ ПРОГРАММЫ	5
1.1 Общие сведения.....	5
1.2 Разработка модуля программы.....	5
1.3 Внесённая ошибка.....	6
2 Разработка документации модуля программы	8
3 Тестирование по	11
4 Исправление ошибки	12
5 Итоговое тестирование	13
ЗАКЛЮЧЕНИЕ	14
Приложения	15

ВВЕДЕНИЕ

Модульное тестирование (или unit-тестирование) — это метод тестирования программного обеспечения, который направлен на проверку отдельных модулей или компонентов программы. Модуль в этом контексте представляет собой небольшую, изолированную часть программы, такую как функция, метод, класс или объект, которая выполняет определенную задачу или функцию в приложении.

Целью модульного тестирования является убеждение в том, что каждый модуль работает правильно и соответствует спецификациям. Этот метод позволяет выявить и исправить ошибки на ранних этапах разработки, что упрощает процесс отладки и обеспечивает более надежное программное обеспечение.

Основные принципы модульного тестирования:

1. **Изоляция:** Каждый модуль тестируется в изоляции от остальных частей программы, чтобы убедиться, что результаты тестирования зависят только от данного модуля.
2. **Автоматизация:** Тесты следует автоматизировать, чтобы можно было легко повторять их выполнение при каждом изменении кода.
3. **Маленькие объемы:** Тесты должны быть относительно маленькими и проверять только ограниченный набор функциональности модуля.
4. **Покрытие:** Важно обеспечить хорошее покрытие кода тестами, чтобы убедиться, что каждая часть модуля проверяется.
5. **Независимость:** Тесты должны быть независимыми друг от друга, чтобы их можно было выполнять в любом порядке и изменять без воздействия на другие тесты.

ПОСТАНОВКА ЗАДАЧИ

Цель работы: познакомиться с концепцией модульного тестирования, научиться проектировать и реализовывать модульные тесты для отдельных компонентов программного обеспечения.

Задачи:

- Изучить основы модульного тестирования.
- Освоить инструменты для написания и выполнения модульных тестов (например, JUnit для Java или pytest для Python).
- Написать модульные тесты для предоставленного примера программы.
- Проанализировать результаты выполнения тестов, выявить и исправить ошибки в коде, если они имеются.

Оборудование и ПО:

- Компьютер с установленной средой разработки (IDE) и выбранным языком программирования.
- Библиотеки для модульного тестирования (при необходимости).
- Любые другие инструменты для тестирования в зависимости от языка программирования.

1 СОЗДАНИЕ МОДУЛЯ ПРОГРАММЫ

1.1 Общие сведения

Используемый язык программирования – Python.

Инструмент для модульного тестирования – библиотека pytest.

В качестве программного продукта было разработано консольное приложение, которое позволяет пользователю сгенерировать пароль по заданным параметрам, а также принять участие в таких мини-играх, как устная или письменная проверка знания технической терминологии на английском языке и быстрый счёт.

Программу можно условно разделить на четыре основных модуля, за разработку каждого из которых отвечал отдельный член команды. В целях удобства тестирование проводилось попарно. В таблице 1.1.1 представлены названия основных модулей программы, а также указан разработчик и тестировщик для каждого модуля.

Таблица 1.1.1 – Модули программы

Название модуля	Разработчик	Тестировщик
Модуль генерации слов для игры	Егоров Л.А.	Корольков А.Д.
Модуль генерации паролей	Корольков А.Д.	Егоров Л.А.
Модуль проверки ввода	Преснякова А.В.	Кликушин В.И.
Модуль «Умный счёт»	Кликушин В.И.	Преснякова А.В.

1.2 Разработка модуля программы

Создан модуль программы, отвечающий за реализацию мини-игры «Проверка знаний английского языка». Пользователю предложено консольное меню, где, после ввода нужной цифры (1) запускается соответствующая

активность. Модуль состоит из функций, отвечающих за считывание англо-русского словаря из файла, считывания параметров игры от пользователя и запуска игрового цикла. Код модуля представлен в Приложение А.

1.3 Внесённая ошибка

В функции, вычисляющей генерирующей тест для перевода английских слов на русский, была допущена ошибка, которая заключается в том, что функция добавляет в тест в два раза меньше вопросов. Обнаружить ошибку можно, запустив игру «Слова» и дождавшись прохождения половины теста – тогда ошибка обязательно возникает. В ином случае ошибка появляется крайне редко. Код функции «generate_en_word_list», в которой допущена ошибка, представлен на рисунке 1.3.1, пример появления ошибки представлен на Рисунке 1.3.2.

```
def generate_en_word_list(cnt: int, word_dictionary: list[dict[str, str]]) -> list[str]:  
    english_words = [tuple(elem.items())[0] for elem in word_dictionary]  
    random.shuffle(english_words)  
    english_words = english_words[:cnt // 2]  
    return english_words
```

Рисунок 1.3.1 – Код функции «generate_en_word_list»

```

PS D:\Study\Testing> python .\CardHub.py
Введите режим игры (1 - 'слова' или 2 - 'быстрый счёт' или 3 - 'генерация пароля'): 1
Введите количество слов 6
Введите, будет ли показываться перевод: Да/нет: нет
Введите количество вариантов ответа 3
Введите, будет ли тест письменным : Да/нет: нет
Введите время на раздумья в секундах 1
фишинг, интернет-мошенничество с целью хищения личных данных, выуживание паролей
backup
ROM read-only memory
phishing
с выходом в интернет (с поддержкой веб-доступа)
keylogger
web-enabled devices
mainframe
язык ассемблера
distortion
assembly language
trigger
Traceback (most recent call last):
  File "D:\Study\Testing\CardHub.py", line 561, in <module>
    main()
  File "D:\Study\Testing\CardHub.py", line 557, in main
    set_mode(big_data)
  File "D:\Study\Testing\CardHub.py", line 50, in set_mode
    glossary_test(big_data)
  File "D:\Study\Testing\CardHub.py", line 202, in glossary_test
    print(loop_oral_test(big_data, word_list, *data))
  File "D:\Study\Testing\CardHub.py", line 80, in loop_oral_test
    return loop_oral_test(big_data, word_list, num_words - 1, show_translation, alternatives, time_to_think, start_num_words)
  File "D:\Study\Testing\CardHub.py", line 80, in loop_oral_test
    return loop_oral_test(big_data, word_list, num_words - 1, show_translation, alternatives, time_to_think, start_num_words)
  File "D:\Study\Testing\CardHub.py", line 80, in loop_oral_test
    return loop_oral_test(big_data, word_list, num_words - 1, show_translation, alternatives, time_to_think, start_num_words)
  File "D:\Study\Testing\CardHub.py", line 68, in loop_oral_test
    key, value = word_list[start_num_words - num_words]
IndexError: list index out of range

```

Рисунок 1.3.2 – Пример возникновения ошибки

2 РАЗРАБОТКА ДОКУМЕНТАЦИИ МОДУЛЯ ПРОГРАММЫ

Модуль представляет из себя набор функций, которые в совокупности реализуют мини-игру «Проверка знаний английского языка». Для удобства прописана аннотация типов, позволяющая лучше понять, какие данные передаются между функциями.

Пользователю предложено ввести количество вопросов в тесте – слова, знание которых должен продемонстрировать пользователь. После этого пользователю предлагается ввести необходимость показа правильного ответа, количество вариантов ответа в каждом вопросе, необходимость вручную вписывать ответ на вопрос и время на размышление для ответа.

Функции:

- **set dictionary**

Параметры: filename: str – путь до файла, содержащего англо-русский словарь.

Тип возвращаемого значения: list[dict] – список из словарей, содержащих в качестве ключа английское слово, а в качестве значения – перевод на русский язык

Описание: Функция считывает словарь из файла и переводит его в список словарей, с которым работает программа.

- **generate en word list**

Параметры: cnt: int – количество вопросов в генерируемом тесте, big_data: list[dict] – словарь, на основе которого производится генерация.

Тип возвращаемого значения: list[tuple].

Описание: Функция возвращает список из пар «английское слово» - «русское слово»

- **generate ru word list**

Параметры: cnt: int – количество вопросов в генерируемом тесте, big_data: list[dict] – словарь, на основе которого производится генерация.

Тип возвращаемого значения: list[tuple].

Описание: Функция возвращает список из пар «русское слово» - «английское слово»

- **set mode**

- **Параметры:** big_data: list[dict] – англо-русский словарь.

Тип возвращаемого значения: bool – проверка корректности ввода.

Описание: функция просит пользователя ввести режим игры и запускает нужную функцию.

- **loop oral test**

Параметры: big_data: list – англо-русский словарь, word_list: список вопросов для теста, num_words: int – количество вопросов, show_translation: bool – показывать перевод или нет, alternatives: int – количество вариантов ответа, time_to_think: int – время на размышление, start_num_words: int – начальное количество вопросов в тесте.

Тип возвращаемого значения: str – итоговая строка результата.

Описание: Рекурсивная функция, которая проводит тест, постепенно показывая новые вопросы и показывая ответы на них.

- **loop written test**

Параметры: big_data: list – англо-русский словарь, word_list: список вопросов для теста, num_words: int – количество вопросов, show_translation: bool – показывать перевод или нет, alternatives: int – количество вариантов ответа, time_to_think: int – время на размышление, start_num_words: int – начальное количество вопросов в тесте, correct: int – количество правильных ответов на

данном этапе.

Тип возвращаемого значения: str – итоговая строка результата.

Описание: Рекурсивная функция, которая проводит тест, постепенно показывая новые вопросы и показывая ответы на них.

3 ТЕСТИРОВАНИЕ ПО

Проведено тестирование в полном объёме модуля генерации паролей, написанного другим членом команды. Для основных функций написаны модульные тесты, которые сравнивают ожидаемый результат с фактическим. Оригинальные функции модуля проверки генерации паролей представлены в приложении Б. Написанный класс TestPassword, реализующий тестирование, представлен в Приложении В. Результат тестирования показан на Рисунке 3.1.

```
PS D:\Study\Testing> pytest .\CardHub.py
===== test session starts =====
platform win32 -- Python 3.10.0, pytest-8.3.3, pluggy-1.5.0
rootdir: D:\Study\Testing
collected 5 items

CardHub.py ..F..

===== FAILURES =====
_____ TestPassword.test_generate_password _____

self = <CardHub.TestPassword object at 0x0000021CE0BC95A0>, mock_digits = <MagicMock name='input_special_symbols' id='2323052801056'>, mock_lower = <MagicMock name='input_uppers' id='2323052801056'>, mock_upper = <MagicMock name='input_lower' id='2323052801056'>, mock_special = <MagicMock name='input_digits' id='2323053485728'>

    @mock.patch('CardHub.input_digits', return_value=True)
    @mock.patch('CardHub.input_lower', return_value=True)
    @mock.patch('CardHub.input_upper', return_value=True)
    @mock.patch('CardHub.input_special_symbols', return_value=False)
    def test_generate_password(self: object, mock_digits: bool, mock_lower: bool, mock_upper: bool, mock_special: bool):
        for i in range(100):
            l = random.randint(6, 12)
            with mock.patch('CardHub.input_length', return_value=l):
                > assert len(generate_password()) == l
                E       AssertionError: assert 5 == 6
                E         + where 5 = len('')xts'
                E         + where '')xts' = generate_password()

CardHub.py:385: AssertionError

===== short test summary info =====
FAILED CardHub.py::TestPassword::test_generate_password - AssertionError: assert 5 == 6
===== 1 failed, 4 passed in 0.51s =====
```

Рисунок 3.1 – Результат тестирования функций модуля проверки генерации паролей

На рисунке видно, что были пройдены не все тесты. Была допущена ошибка в написании функции «test_generate_password». Информация о данной ошибке передана разработчику для дальнейшего исправления.

4 ИСПРАВЛЕНИЕ ОШИБКИ

Краткое описание ошибки: Длина сгенерированного пароля может оказаться меньше запрошенной пользователем

Статус ошибки: открыта («Open»).

Категория ошибки: серьёзная («Major»).

Тестовый случай: «Проверка длины генерируемого пароля».

Описание ошибки:

1. Запустить программу.
2. Вызвать функцию `generate_password`, симулировав ввод различных параметров с клавиатуры.
3. Полученный результат: неправильная длина пароля в некоторых случаях.

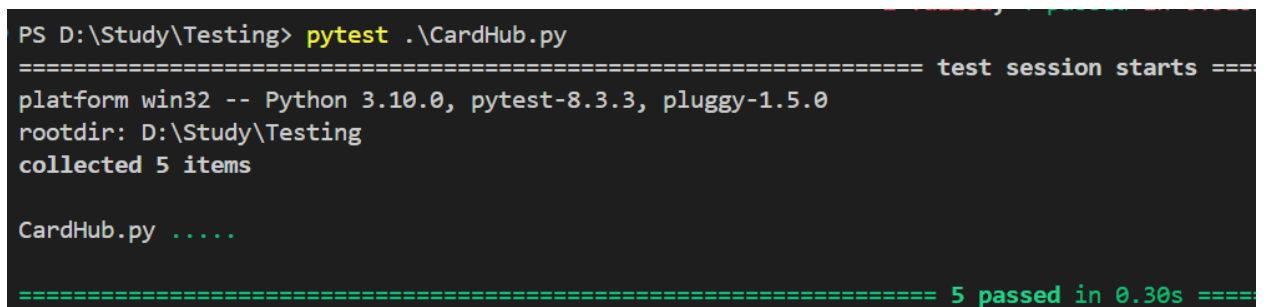
Ожидаемый результат: длина пароля в каждом случае генерации должна совпадать с ожидаемой.

Созданная документация на ошибку передаётся обратно разработчику ПО для её исправления.

5 ИТОГОВОЕ ТЕСТИРОВАНИЕ

После исправления ошибки, модуль программы был возвращен на повторную проверку. Так, было проведено повторное тестирование, в результате которого все Unit-тесты были успешно пройдены.

На Рисунке 5.1 представлен результат успешного прохождения всех тестов.



```
PS D:\Study\Testing> pytest .\CardHub.py
===== test session starts =====
platform win32 -- Python 3.10.0, pytest-8.3.3, pluggy-1.5.0
rootdir: D:\Study\Testing
collected 5 items

CardHub.py .....
===== 5 passed in 0.30s =====
```

Рисунок 5.1 – Результат прохождения всех Unit-тестов

ЗАКЛЮЧЕНИЕ

В ходе выполнения практической работы были изучены ключевые аспекты модульного тестирования, а также освоены инструменты, предназначенные для создания и выполнения тестов на языке программирования Python, в частности библиотека Pytest. В рамках проекта были разработаны модульные тесты для программы, которая осуществляет операции с физическими величинами, что позволило проверить корректность её работы. По результатам анализа тестов были обнаружены несколько ошибок в исходном коде программы, которые могли приводить к некорректным вычислениям. Выявленные проблемы были успешно исправлены другим членом команды, что позволило улучшить качество программы и добиться её стабильного функционирования.

ПРИЛОЖЕНИЯ

Приложение А – разработанные функции

Приложение Б – функции другого члена команды

Приложение В – функции для тестирования

Приложение А

Листинг А.1 – Разработанные функции

```
def set_dictionary(filename: str) -> list[dict]:
    '''Функция для формирования "словаря" из файла с заданным именем'''
    dictionary_data = list()
    try:
        with open(filename, 'r', encoding='utf-8') as file:
            for line in file:
                en_word, ru_word = line.split('==')
                dictionary_data.append({en_word.strip(): ru_word.strip()})
    except FileNotFoundError:
        pass
    return dictionary_data

def generate_en_word_list(cnt: int, word_dictionary: list[dict[str, str]])
-> list[str]:
    english_words = [tuple(elem.items())[0] for elem in word_dictionary]
    random.shuffle(english_words)
    english_words = english_words[:cnt // 2]
    return english_words

def generate_ru_word_list(cnt: int, word_dictionary: list[dict[str, str]])
-> list[str]:
    russian_words = [tuple(elem.items())[0] for elem in word_dictionary]
    random.shuffle(russian_words)
    russian_words = russian_words[:cnt]
    return russian_words

def set_mode(big_data: dict) -> bool:
    mode = int(input(
        "Введите режим игры (1 - 'слова' или 2 - 'быстрый счёт' или 3 -
        'генерация пароля'): "))
    match mode:
        case 1:
            glossary_test(big_data)
        case 2:
            set_fast_count()
        case 3:
            password = generate_password()
            if password is None:
                print('Пароль не получилось сгенерировать из-за длины
                пароля')
            else:
                print(password)
        case _:
            return False
    return mode in [1, 2, 3]

def loop_oral_test(big_data: list, word_list: list, num_words: int,
show_translation: bool, alternatives: int, time_to_think: int,
start_num_words: int) -> str:
    answers = list()
    if num_words == 0:
        return "Количество новых изученных слов - {}, потраченное время - {}
секунд".format(start_num_words, start_num_words*(time_to_think+1))
    key, value = word_list[start_num_words - num_words]
    for _ in range(alternatives - 1):
        wrong_key, _ = list(random.choice(big_data).items())[0]
        answers.append(wrong_key)
    answers.append(key)
    random.shuffle(answers)
```



```
print('\033[103m' + str(value) + '\033[0m')
print(*answers, sep='\n')
time.sleep(time_to_think)
if show_translation:
    print('Правильный ответ: \033[102m' + str(key) + '\033[0m\n')
    time.sleep(1)
return loop_oral_test(big_data, word_list, num_words - 1,
show_translation, alternatives, time_to_think, start_num_words)
```

Приложение Б

Листинг Б.1 – Функции другого члена команды

```
def password_alphabet(digits: bool, lowers: bool, uppers: bool,
special_symbols: bool) -> list:
    alphabet = [':', ')', '(']
    if digits:
        alphabet.extend(string.digits)
    if lowers:
        alphabet.extend(string.ascii_lowercase)
    if uppers:
        alphabet.extend(string.ascii_uppercase)
    if special_symbols:
        alphabet.extend('!#$%&*+~=?@^_')
    return alphabet

def check_password_length(password_length: int, conditions: int) -> bool:
    return conditions <= password_length and password_length > 0

def generate_password() -> str | None:
    result = password_settings()
    if result is not None:
        length, digits, lowers, uppers, special_symbols = result
        alphabet = password_alphabet(digits, lowers, uppers, special_symbols)
        password = str()
        for _ in range(length - 1):
            password += random.choice(alphabet)
        return password

def input_length() -> int:
    length = int(input('Введите длину пароля: '))
    return length

def input_uppers() -> bool:
    uppers = input('Включать ли заглавные буквы? (Впишите + / -): ')
    uppers = True if uppers == '+' else False
    return uppers
```

Приложение В

Листинг В.1 – Функции для тестирования

```
class TestPassword:
    def test_password_alphabet(self: object) -> None:
        for param in product([False, True], repeat=4):
            digits, lowers, uppers, special_symbols = param
            alphabet = password_alphabet(*param)
            if digits:
                assert all([digit in alphabet for digit in string.digits])
                assert len(alphabet) >= len(string.digits)
            if lowers:
                assert all(
                    [lower in alphabet for lower in string.ascii_lowercase])
                assert len(alphabet) >= len(string.ascii_lowercase)
            if uppers:
                assert all(
                    [uppers in alphabet for uppers in
string.ascii_uppercase])
                assert len(alphabet) >= len(string.ascii_uppercase)
            if special_symbols:
                assert all([sp in alphabet for sp in '!#$%&*+~=?@^_'])
                assert len(alphabet) >= len('!#$%&*+~=?@^_')
            if all([not elem for elem in [digits, lowers, uppers,
special_symbols]]):
                assert len(alphabet) == 3

    def test_check_password_length(self: object) -> None:
        password_lengths = [37, 59, 2, 12, 7, -5, 0, 1]
        conditions = [3, 4, 3, 1, 0, 3, 3, 4]
        answers = [True, True, False, True, True, False, False, False]
        for passw, cond, ans in zip(password_lengths, conditions, answers):
            assert check_password_length(passw, cond) == ans

    @mock.patch('CardHub.input_digits', return_value=True)
    @mock.patch('CardHub.input_lowes', return_value=True)
    @mock.patch('CardHub.input_uppers', return_value=True)
    @mock.patch('CardHub.input_special_symbols', return_value=False)
    def test_generate_password(self: object, mock_digits: bool, mock_lowes:
bool, mock_uppers: bool, mock_specials: bool) -> None:
        for i in range(100):
            l = random.randint(6, 12)
            with mock.patch('CardHub.input_length', return_value=l):
                assert len(generate_password()) == l

    def test_input_length(self: object) -> None:
        input_generator = (answ for answ in
['1','3','4','5','0','7','8','9'])
        for answ in input_generator:
            with mock.patch('builtins.input', lambda _: answ):
                num = input_length()
                assert num == int(answ)

    def test_input_uppers(self: object) -> None:
        input_generator = (answ for answ in ['+', '-', '-', '+', 'f'])
        ans = [True, False, False, True, False]
        for answ,a in zip(input_generator,ans):
            with mock.patch('builtins.input', lambda _: answ):
                upp = input_uppers()
                assert upp == a
```