МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное автономное образовательное учреждение
высшего образования «Санкт-Петербургский политехнический
университет Петра Великого»

Институт компьютерных наук и кибербезопасности

Высшая школа технологий искусственного интеллекта

Направление: 02.03.01 «Математика и компьютерные науки»

Лабораторная работа №3
«Статический анализ кода приложений»
по дисциплине
«Методы тестирования программного обеспечения»

Студент,
группы 5130201/20102 _____ Тищенко А. А.

Преподаватель _____ Курочкин М. А.

«_____»_____ 2025г.

Санкт-Петербург, 2025

Содержание

1 Постановка задачи

Задачи лабораторной работы:

- изучить методы статического тестирование;
- провести статическое тестирование программы;
- проанализировать полученный результат;
- рассмотреть рекомендации статического анализатора и при необходимости внести изменения в программу.

2 Статическое тестирование

Статическое тестирование — это оценка элемента тестирования, при которой не происходит выполнения кода, и которая может быть проведена вручную или с помощью инструментариев. Объектом тестирования может быть документация или исходный код, а сам процесс возможен на любом этапе жизненного цикла ПО (Согласно ГОСТ Р 56920-2024 (раздел 5.5.2)).

Статическое тестирование включает в себя:

- Проверку документации: требования, спецификации, архитектурные решения.
- Анализ исходного кода: поиск синтаксических ошибок, нарушений стандартов кодирования и потенциальных уязвимостей.

Цель статического тестирования — выявление дефектов на ранних стадиях разработки, что снижает затраты на их исправление. Как отмечает Гленфорд Майерс в книге «Искусство тестирования», до 60% ошибок можно обнаружить до запуска программы. Это делает статическое тестирование критически важным инструментом для повышения качества ПО.

2.1 Основные формы статического тестирования

Используются специальные инструменты – «статические анализаторы», которые автоматически:

- Проверяют соответствие кода стандартам кодирования (Code Style, Coding Guidelines).
- Ищут потенциальные ошибки (например, неиспользуемые переменные, некорректные приведения типов, опасные конструкции).
- Указывают на потенциальные уязвимости в безопасности (например, возможности для SQL-инъекций, потенциальные переполнения буфера).
- Анализируют потоки данных, чтобы понять, где значения могут принимать нежелательные (NullPointerException и др.) значения.

2.2 Разница между статическим анализатором и инспекцией кода за столом

Способ выполнения

- Статический анализатор: запускается автоматически на исходном коде и выдает отчёт об обнаруженных проблемах. Анализатор следует набору заранее заданных правил (линейный и/или межпроцедурный анализ, анализ потока данных и т. д.).

- Инспекция кода за столом: проводится людьми (разработчиками, тестировщиками). Участники встречи просматривают код построчно (или анализируют его логические куски) и обсуждают архитектурные, логические, стилевые и другие аспекты.

Область охвата

- Статический анализатор:
  - Ориентирован в основном на типичные ошибки и «сигнализирует» о потенциальных проблемах, отклонениях от правил кодирования, уязвимостях в безопасности.
  - Хорошо справляется с рутинным поиском большого количества распространённых проблем (например, неиспользуемые переменные, неочевидные «if» без «else», выход за границы массива и т. п.).
- Инспекция кода:
  - Позволяет вскрыть более сложные логические ошибки, несоответствие требованиям, некорректную бизнес-логику.
  - Во время обсуждения могут выявиться проблемы, которые невозможно уловить статическим анализатором: «Почему этот алгоритм выбран именно так?», «Соответствует ли это бизнес-требованиям?», «Оптимальна ли структура данных?», «Легко ли будет поддерживать этот код?».

Глубина и виды обнаруживаемых дефектов

- Статический анализатор: находит скорее «структурные» и «синтаксические» дефекты и уязвимости (неиспользуемые переменные, неправильные операции с памятью, отсутствие проверок). Может не понимать, «хорош ли» сам алгоритм.
- Инспекция кода: ориентирована на логику, архитектуру, читаемость, потенциальные проблемы взаимодействия модулей. Тут важны не только дефекты самого кода, но и соответствует ли он требованиям или лучшим практикам проектирования.

Скорость и автоматизация

- Статический анализатор: работает быстро (особенно если хорошо интегрирован в CI/CD); выдаёт отчёты сразу после запуска.
- Инспекция кода: процесс требует участия людей и времени на обсуждение. Однако именно в этом процессе выявляются «глубинные» проблемы, которые не найдёт автоматизированный инструмент.

Результаты и интерпретация

- Статический анализатор:
  - Даёт отчёты (логи, списки ошибок/предупреждений) – но они нуждаются в интерпретации человеком, поскольку есть ложные срабатывания (false positives).
  - Для принятия решения о серьёзности проблемы часто всё равно приходится просматривать код.
- Инспекция кода:
  - Часто приводит не только к обнаружению ошибок, но и к улучшению совместной экспертизы в команде.
  - Может завершиться рекомендациями по рефакторингу, изменению архитектуры, или даже пересмотром требований.

3 Описание приложения и среды разработки

Название программы: Генератор паролей.

Задача программы: Сгенерировать пароль с параметрами, заданными пользователем.

Дано:

- число – длина генерируемого пароля;
- строка «yes» или «no», если пользователь введёт строку «yes>, то в пароле будут использоваться строчные буквы;
- строка «yes» или «no», если пользователь введёт строку «yes>, то в пароле будут использоваться заглавные буквы;
- строка «yes» или «no», если пользователь введёт строку «yes>, то в пароле будут использоваться цифры;
- строка «yes» или «no», если пользователь введёт строку «yes>, то в пароле будут использоваться спецсимволы;
- число – минимальное количество строчных букв в пароле;
- число – минимальное количество заглавных букв в пароле;
- число – минимальное количество цифр в пароле;
- число – минимальное количество спецсимволов;

Ограничения:

- допустимая длина пароля – от 1 до 100 символов;
- минимальное количество строчных букв – 0;
- минимальное количество заглавных букв – 0;
- минимальное количество цифр – 0;
- минимальное количество спецсимволов – 0;
- сумма минимального количества строчных букв, заглавных букв, цифр и спецсимволов не может превышать длину пароля.

Программа была написана на языке Python. В качестве среды разработки использовалось лицензионное программное обеспечение для редактирования текста – Microsoft Visual Studio Code.

3.1 Спецификация тестируемой программы

Спецификация программы представлена в таблице 1.

Таблица 1. Спецификация.

Входные данные	**Выходные данные**	**Комментарий**
10 y y y y 1 1 1 1	6KoL4Tfn*M	Программа сгенерировала и вывела на экран пароль с заданными параметрами.
-10 y y y y 1 1 1 1	Ошибка: значение должно быть не меньше 1. Попробуйте снова.	Программа вывела на экран сообщение о некорректном пользовательском вводе.
10 abc y y y 1 1 1 1	Ошибка: введите 'yes' (или 'y') или 'no' (или 'n').	Программа вывела на экран сообщение о некорректном пользовательском вводе.
10 abc y y y -1 1 1 1	Ошибка: значение должно быть не меньше 0. Попробуйте снова.	Программа вывела на экран сообщение о некорректном пользовательском вводе.
10 abc y y y 5 5 5 5	Ошибка: сумма минимальных значений (103) превышает длину пароля (10). Пожалуйста, введите параметры заново.	Программа вывела на экран сообщение о некорректном пользовательском вводе.

3.2 Исходный код программы тестируемой программы

Исходный код программы представлен в листинге 1.

Листинг 1. Исходный код.

```
import random
import string

max_password_length = 100  # Максимальная длина пароля


def get_valid_int(prompt, min_value=0, max_value=None):
    while True:
        user_input = input(prompt).strip()
        if not user_input:
            print("Ошибка: ввод не должен быть пустым. Попробуйте снова.")
            continue
        try:
            value = int(user_input)
            if value < min_value:
                print(
                    f"Ошибка: значение должно быть не меньше {min_value}. Попробуйте снова."
                )
```

```
19                 continue
20             if max_value and value > max_value:
21                 print(
22                     f"Ошибка: значение должно быть не больше {max_value}. Попробуйте снова."
23                 )
24                 continue
25             return value
26         except ValueError:
27             print("Ошибка: введите корректное целое число.")
28
29
30 def get_yes_no(prompt):
31     while True:
32         user_input = input(prompt).strip().lower()
33         if user_input in ["yes", "y"]:
34             return True
35         if user_input in ["no", "n"]:
36             return False
37         print("Ошибка: введите 'yes' или ('y') или 'no' или ('n').")
38
39
40
41 def get_user_input():
42     """Запрашивает у пользователя параметры генерации пароля с проверкой ввода."""
43     global max_password_length
44
45     length = get_valid_int(
46         f"Введите длину пароля (1-{max_password_length}): ",
47         min_value=1,
48         max_value=max_password_length,
49     )
50
51     use_lower = get_yes_no("Использовать строчные буквы? (yes/y, no/n): ")
52     use_upper = get_yes_no("Использовать заглавные буквы? (yes/y, no/n): ")
53     use_digits = get_yes_no("Использовать цифры? (yes/y, no/n): ")
54     use_special = get_yes_no("Использовать спецсимволы (!@#$%^&*)? (yes/y, no/n): ")
55
56     # Проверяем, что хотя бы один тип символов выбран
57     if not (use_lower or use_upper or use_digits or use_special):
58         print("Ошибка: необходимо выбрать хотя бы один тип символов.")
59         return get_user_input()  # Повторный ввод всех данных
60
61     # Запрашиваем минимальное количество каждого типа символов
62     min_lower = (
63         get_valid_int("Минимальное количество строчных букв: ", 0) if use_lower else 0
64     )
65     min_upper = (
66         get_valid_int("Минимальное количество заглавных букв: ", 0) if use_upper else 0
67     )
68     min_digits = get_valid_int("Минимальное количество цифр: ", 0) if use_digits else 0
69     min_special = (
70         get_valid_int("Минимальное количество спецсимволов: ", 0) if use_special else 0
71     )
72
73     return (
74         length,
```

```
75          use_lower,
76          use_upper,
77          use_digits,
78          use_special,
79          min_lower,
80          min_upper,
81          min_digits,
82          min_special,
83      )
84
85
86  def validate_input(length, min_lower, min_upper, min_digits, min_special):
87      """Проверяет, что длина пароля больше суммы минимальных значений."""
88      total_required = min_lower + min_upper + min_digits + min_special
89      if total_required > length:
90          print(
91              f"Ошибка: сумма минимальных значений ({total_required}) превышает длину пароля
      ({length})."
92          )
93          return False
94      return True
95
96
97  def generate_mandatory_chars(
98      min_lower,
99      min_upper,
100     min_digits,
101     min_special,
102     lower_chars,
103     upper_chars,
104     digit_chars,
105     special_chars,
106 ):
107     """Генерирует обязательные символы пароля."""
108     password = (
109         random.choices(lower_chars, k=min_lower)
110         + random.choices(upper_chars, k=min_upper)
111         + random.choices(digit_chars, k=min_digits)
112         + random.choices(special_chars, k=min_special)
113     )
114     return password
115
116
117 def fill_password(password, length, all_chars):
118     """Дополняет пароль случайными символами до нужной длины."""
119     remaining_length = length - len(password)
120     password += random.choices(all_chars, k=remaining_length)
121     return password
122
123
124 def shuffle_password(password):
125     """Перемешивает символы пароля случайным образом."""
126     random.shuffle(password)
127     return "".join(password)
128
129
```

```python
def generate_password(
    length,
    use_lower,
    use_upper,
    use_digits,
    use_special,
    min_lower,
    min_upper,
    min_digits,
    min_special,
):
    """Генерирует пароль с учётом заданных параметров."""
    lower_chars = string.ascii_lowercase if use_lower else ""
    upper_chars = string.ascii_uppercase if use_upper else ""
    digit_chars = string.digits if use_digits else ""
    special_chars = "!@#$%^&*" if use_special else ""

    all_chars = lower_chars + upper_chars + digit_chars + special_chars

    while not validate_input(length, min_lower, min_upper, min_digits, min_special):
        print("Пожалуйста, введите параметры заново.")
        return generate_password(*get_user_input())

    password = generate_mandatory_chars(
        min_lower,
        min_upper,
        min_digits,
        min_special,
        lower_chars,
        upper_chars,
        digit_chars,
        special_chars,
    )
    password = fill_password(password, length, all_chars)
    password = shuffle_password(password)

    return password


def main():
    """Основная функция программы."""
    max_password_length = 100
    user_data = get_user_input()
    password = generate_password(*user_data)
    print("Сгенерированный пароль:", password)


if __name__ == "__main__":
    main()
```

4 Статический анализ кода приложения

4.1 Описание выбранного инструмента для статического анализа кода

В качестве инструмента статического анализа кода проекта был выбран Pylint версии 3.3.6. Pylint - это программа для проверки исходного кода, ошибок и качества для языка программирования Python. Он назван в соответствии с общепринятым в Python соглашением о префиксе «py» и отсылкой к программе lint для программирования на C. Он следует стилю, рекомендованному PEP 8, руководством по стилю Python. Он похож на Pychecker и Pyflakes, но включает в себя следующие функции:

- Проверка длины каждой строки;
- Проверка правильности формирования имен переменных в соответствии со стандартом кодирования проекта;
- Проверка того, что заявленные интерфейсы действительно реализованы.

Pylint классифицирует свои сообщения об ошибках и предупреждениях по категориям, каждая из которых обозначается соответствующим префиксом. Основные виды сообщений следующие

- C (Convention): Сообщения, связанные со стилем оформления кода (например, нарушение соглашений PEP8), именованием переменных, форматированием и т.д.
- R (Refactor): Рекомендации по рефакторингу кода для улучшения читаемости, структуры и поддерживаемости. Эти сообщения помогают улучшить архитектуру кода.
- W (Warning): Предупреждения о потенциальных проблемах, которые могут привести к ошибкам или неожиданному поведению во время выполнения. Например, возможное использование необъявленной переменной.
- E (Error): Ошибки, которые приведут к сбоям выполнения программы, такие как: синтаксические ошибки, вызов функций с неккоректным списком параметров или с неккоректными типами параметров.

4.2 Разбор Pylint правил из группы «Convention»

Эти проверки нацелены на то, чтобы код соответствовал принятым соглашениям о стиле (например, PEP8) и был легко читаемым.

- Именование:
  - Проверяется, чтобы имена классов, функций, переменных, модулей и констант соответствовали принятым стандартам.

  - Например, классы должны именоваться в стиле CamelCase (например, MyClass), а функции и переменные – в snake_case (например, my_function, my_variable).
  - Также проверяются длина имен и их осмысленность, чтобы они точно отражали назначение объекта в коде.

- Стиль оформления:
  - Длина строк: Pylint следит за тем, чтобы строки не превышали установленную длину (обычно 79 или 99 символов в зависимости от конфигурации).
  - Отступы и пробелы: Контролируется корректное использование отступов (обычно 4 пробела), пробелов вокруг операторов, после запятых и т.д.
  - Разбиение на строки: Рекомендуется правильно разбивать длинные выражения или вызовы функций на несколько строк для лучшей читаемости.
  - Пустые строки: Проверяется количество пустых строк между функциями и классами для поддержания визуальной структуры кода.

- Документация:
  - Docstrings: Pylint обращает внимание на наличие строк документации (docstrings) в модулях, классах, функциях и методах.
  - Формат документации: Документация должна быть оформлена согласно принятым стандартам (например, в формате reStructuredText или Google style), чтобы обеспечить понятное описание функционала и параметров.

4.3 Разбор Pylint правил из группы «Refactor»

Эта группа сообщений направлена на улучшение структуры кода, его упрощение и повышение поддерживаемости

- Сложность функций:
  - Цикломатическая сложность: Анализируется количество ветвлений, циклов и условных операторов. Функции с высокой сложностью могут быть трудными для тестирования и отладки.
  - Слишком длинные функции: Если функция слишком большая или содержит множество аргументов, Pylint может рекомендовать её разбить на более мелкие части.

- Дублирование кода:
  - Проверка на повторяющиеся участки кода, что может указывать на возможность объединения логики в одну функцию или класс.

  - Цель – уменьшить количество повторений, чтобы изменение в одной части кода не требовало повторения исправлений в нескольких местах.

- Структурные проблемы:

  - Обнаружение слишком больших классов или методов, которые выполняют сразу несколько задач
  - Рекомендации по разделению ответственности (например, принцип единственной ответственности из SOLID) для улучшения модульности и тестируемости кода.

4.4 Разбор Pylint правил из группы «Warning»

Эта категория охватывает сообщения, которые указывают на потенциальные проблемы, не являющиеся критическими ошибками, но способными привести к неожиданному поведению.

- Неиспользуемые элементы:

  - Переменные: Если переменная объявлена, но не используется, Pylint сообщает об этом, что помогает избежать загромождения кода.
  - Импорты: Неиспользуемые модули или функции, импортированные в начале файла, могут быть отмечены для удаления.
  - Аргументы функций: Иногда функция принимает аргументы, которые не используются в теле, что может быть сигналом к тому, что интерфейс функции следует пересмотреть.

- Подозрительные конструкции:

  - Использование переменных до объявления: Если переменная используется до того, как ей было присвоено значение, это может привести к ошибкам.
  - Использование изменяемых значений по умолчанию: Применение изменяемых объектов (например, списков или словарей) в качестве значений по умолчанию в параметрах функций может привести к неожиданным эффектам.

- Ошибки логики:

  - Порой конструкция кода может быть синтаксически корректной, но её поведение может быть неочевидным или потенциально приводить к логическим ошибкам (например, некорректное сравнение или неверное использование операторов).

4.5 Разбор Pylint правил из группы «Error»

Эти проверки сигнализируют о проблемах, которые приведут к сбоям выполнения программы.

- Синтаксические ошибки:
  - Ошибки в написании кода (например, забытые двоеточия, скобки, неправильное построение конструкции) могут привести к невозможности интерпретировать код.
  - Такие ошибки выявляются на этапе статического анализа до выполнения программы.
- Ошибки времени выполнения:
  - Обращение к несуществующим атрибутам: Если код пытается получить доступ к атрибуту, которого нет у объекта, Pylint укажет на эту проблему.
  - Неправильное использование функций: Например, вызов метода на объекте неподходящего типа или передача неверных аргументов, что в итоге вызовет исключение при выполнении.

5 Процесс тестирования

5.1 Подготовка

Перед использованием PyLint необходимо установить Python и PIP с официального сайта. Затем создать виртуальное окружение с помощью команды `virtualenv venv`.

Чтобы установить PyLint, достаточно выполнить команду `pip install pylint` внутри виртуального окружения.

Для запуска статического анализа достаточно выполнить команду `pylint file.py`, где `file.py` – это название файла с исходным кодом. При запуске PyLint без дополнительных параметров, статический анализатор будет искать все типы ошибок, перечисленные в предыдущем разделе.

5.2 Результат работы анализатора

Полный вывод команды `pylint passgen.py` представлен в листинге 2. Статический анализатор вывел 12 сообщений о различных проблемах в коде. Они связаны с несоответствием стиля именования, отсутствием документации в модуле и функциях, а также слишком длинными строками. Кроме того, есть предупреждения о неинициализированной глобальной переменной, переопределении имени переменной во внутренней области видимости, а также о слишком большом количестве аргументов в функциях. PyLint также даёт общую оценку кода. Для моей программы он вывел оценку 8.48 из 10, что означает, что код в целом неплох, но его можно улучшить.

Листинг 2. Результат выполнения команды `pylint passgen.py`.

```
1  ************* Module passgen
2  passgen.py:91:0: C0301: Line too long (103/100) (line-too-long)
3  passgen.py:1:0: C0114: Missing module docstring (missing-module-docstring)
4  passgen.py:4:0: C0103: Constant name "max_password_length" doesn't conform to UPPER_CASE
       naming style (invalid-name)
5  passgen.py:7:0: C0116: Missing function or method docstring (missing-function-docstring)
6  passgen.py:30:0: C0116: Missing function or method docstring (missing-function-docstring)
7  passgen.py:43:4: W0602: Using global for 'max_password_length' but no assignment is done (
       global-variable-not-assigned)
8  passgen.py:97:0: R0913: Too many arguments (8/5) (too-many-arguments)
9  passgen.py:97:0: R0917: Too many positional arguments (8/5) (too-many-positional-arguments)
10 passgen.py:130:0: R0913: Too many arguments (9/5) (too-many-arguments)
11 passgen.py:130:0: R0917: Too many positional arguments (9/5) (too-many-positional-arguments)
12 passgen.py:171:4: W0621: Redefining name 'max_password_length' from outer scope (line 4) (
       redefined-outer-name)
13 passgen.py:171:4: W0612: Unused variable 'max_password_length' (unused-variable)
14
15 ------------------------------------------------------------------
16 Your code has been rated at 8.48/10 (previous run: 8.48/10, +0.00)
```

5.3 Результат улучшения кода

В соответствии с рекомендациями PyLint в исходный код были добавлены комментарии с документацией для всего модуля и функций, слишком длинные строки были разбиты на более короткие и удобочитаемые, неинициализированная глобальная переменная была удалена, была удалена одна переменная из локальной области видимости, перекрывавшая глобальную переменную, имена всех переменных были приведены в соответствие с принятыми соглашениями языка Python.

После внесения перечисленных выше изменений статический анализатор PyLint был запущен ещё раз на обновлённом файле с исходным кодом. Результат запуска представлен в листинге 3. В этот раз PyLint не вывел никаких сообщений.

Листинг 3. Результат работы PyLint на обновлённом файле с исходным кодом.

```
1 ------------------------------------
2 Your code has been rated at 10.00/10
```

Обновлённый код программы представлен в листинге 4.

Листинг 4. Обновлённый код программы.

```
1  """Модуль
2   для генерации безопасных паролей с разными настройками.Пользователь
3   может задавать длину, типы символов и минимальное количество каждого типа.
4  """
5
6  import random
7  import string
8
9  MAX_PASSWORD_LENGTH = 100  # Максимальная длина пароля
10
11
12 def get_valid_int(prompt, min_value=0, max_value=None):
13     """Запрашивает у пользователя целое число, проверяя корректность ввода."""
14     while True:
15         user_input = input(prompt).strip()
16         if not user_input:
17             print("Ошибка: ввод не должен быть пустым. Попробуйте снова.")
18             continue
19         try:
20             value = int(user_input)
21             if value < min_value:
22                 print(
23                     f"Ошибка: значение должно быть не меньше {min_value}. Попробуйте снова."
24                 )
25                 continue
26             if max_value and value > max_value:
27                 print(
28                     f"Ошибка: значение должно быть не больше {max_value}. Попробуйте снова."
29                 )
30                 continue
31             return value
32         except ValueError:
33             print("Ошибка: введите корректное целое число.")
34
35
```

```
36  def get_yes_no(prompt):
37      """Запрашивает у пользователя 'yes'/'y' или 'no'/'n', проверяя корректность ввода."""
38      while True:
39          user_input = input(prompt).strip().lower()
40          if user_input in ["yes", "y"]:
41              return True
42          if user_input in ["no", "n"]:
43              return False
44          print("Ошибка: введите 'yes' или ('y') или 'no' или ('n').")
45
46
47  def get_user_input():
48      """Запрашивает у пользователя параметры генерации пароля с проверкой ввода."""
49      settings = {
50          "length": get_valid_int(
51              f"Введите длину пароля (1-{MAX_PASSWORD_LENGTH}): ",
52              min_value=1,
53              max_value=MAX_PASSWORD_LENGTH,
54          ),
55          "use_lower": get_yes_no("Использовать строчные буквы? (yes/y, no/n): "),
56          "use_upper": get_yes_no("Использовать заглавные буквы? (yes/y, no/n): "),
57          "use_digits": get_yes_no("Использовать цифры? (yes/y, no/n): "),
58          "use_special": get_yes_no(
59              "Использовать спецсимволы (!@#$%^&*)? (yes/y, no/n): "
60          ),
61      }
62
63      # Проверяем, что хотя бы один тип символов выбран
64      if not any(
65          [
66              settings["use_lower"],
67              settings["use_upper"],
68              settings["use_digits"],
69              settings["use_special"],
70          ]
71      ):
72          print("Ошибка: необходимо выбрать хотя бы один тип символов.")
73          return get_user_input()  # Повторный ввод всех данных
74
75      # Запрашиваем минимальное количество каждого типа символов
76      settings["min_lower"] = (
77          get_valid_int("Минимальное количество строчных букв: ", 0)
78          if settings["use_lower"]
79          else 0
80      )
81      settings["min_upper"] = (
82          get_valid_int("Минимальное количество заглавных букв: ", 0)
83          if settings["use_upper"]
84          else 0
85      )
86      settings["min_digits"] = (
87          get_valid_int("Минимальное количество цифр: ", 0)
88          if settings["use_digits"]
89          else 0
90      )
91      settings["min_special"] = (
```

```
92          get_valid_int("Минимальное количество спецсимволов: ", 0)
93          if settings["use_special"]
94          else 0
95      )
96
97      return settings
98
99
100 def validate_input(settings):
101     """Проверяет, что длина пароля больше суммы минимальных значений."""
102     total_required = sum(
103         [
104             settings["min_lower"],
105             settings["min_upper"],
106             settings["min_digits"],
107             settings["min_special"],
108         ]
109     )
110     if total_required > settings["length"]:
111         print(
112             f"Ошибка: сумма минимальных значений ({total_required})"
113             f" превышает длину пароля ({settings['length']})."
114         )
115         return False
116     return True
117
118
119 def generate_mandatory_chars(settings, char_sets):
120     """Генерирует обязательные символы пароля."""
121     password = (
122         random.choices(char_sets["lower"], k=settings["min_lower"])
123         + random.choices(char_sets["upper"], k=settings["min_upper"])
124         + random.choices(char_sets["digits"], k=settings["min_digits"])
125         + random.choices(char_sets["special"], k=settings["min_special"])
126     )
127     return password
128
129
130 def fill_password(password, length, all_chars):
131     """Дополняет пароль случайными символами до нужной длины."""
132     remaining_length = length - len(password)
133     password += random.choices(all_chars, k=remaining_length)
134     return password
135
136
137 def shuffle_password(password):
138     """Перемешивает символы пароля случайным образом."""
139     random.shuffle(password)
140     return "".join(password)
141
142
143 def generate_password(settings):
144     """Генерирует пароль с учётом заданных параметров."""
145     char_sets = {
146         "lower": string.ascii_lowercase if settings["use_lower"] else "",
147         "upper": string.ascii_uppercase if settings["use_upper"] else "",
```

```
148         "digits": string.digits if settings["use_digits"] else "",
149         "special": "!@#$%^&*" if settings["use_special"] else "",
150     }
151
152     all_chars = "".join(char_sets.values())
153
154     while not validate_input(settings):
155         print("Пожалуйста, введите параметры заново.")
156         return generate_password(get_user_input())
157
158     password = generate_mandatory_chars(settings, char_sets)
159     password = fill_password(password, settings["length"], all_chars)
160     password = shuffle_password(password)
161
162     return password
163
164
165 def main():
166     """Основная функция программы."""
167     user_settings = get_user_input()
168     password = generate_password(user_settings)
169     print("Сгенерированный пароль:", password)
170
171
172 if __name__ == "__main__":
173     main()
```

Заключение

В ходе выполнения данной лабораторной работы было проведено статистическое тестирование для программы: «Генератор паролей».

Были найдены следующие проблемы:

- 5 нарушений правил оформления исходного кода;
- 3 предупреждения о возможных ошибках в исходном коде;
- 4 рекомендации по рефакторингу исходного кода.

В результате проделанной работы программный код был исправлен в соответствии с рекомендациями статистического анализатора. На примере небольшой программы была наглядно продемонстрирована польза от использования статических анализаторов кода. Были сделаны выводы о том, что статистическое тестирование позволяет выявить некорректность в первую очередь в стиле оформления программы. На примере PyLint был получен первый опыт использования статических анализаторов кода.

В процессе статистического тестирования были выявлены некоторые недостатки, которых не удалось обнаружить во время инспекции за столом. В основно они связаны с оформлением кода. Например, при инспекции кода не было замечаний по поводу отсутствующих комментариев с документацией. Также была пропущена достаточно серьёзная неточность – переменная из внешней области видимости перекрывалась переменной из локальной области видимости. Однако во время инспекции была обнаружена логическая ошибка при формировании пароля, состоящего только из заглавных символов. На основе полученных результатов можно сделать вывод, что статистическое тестирование является хорошим дополнением к инспекции за столом. К тому же статическое тестирование не столь трудозатрано и его можно автоматизировать.

Список литературы

[1] Майерс, Г. Искусство тестирования программ. – Санкт-Петербург: Диалектика, 2012 г.