МИНОБРНАУКИ РОССИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

«САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО»

Институт компьютерных наук и кибербезопасности

Направление 02.03.01 Математика и компьютерные науки

ОТЧЕТ ПО ДИСЦИПЛИНЕ «МЕТОДЫ ТЕСТИРОВАНИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ»

Лабораторная работа №1 «Инспекция кода»

Обучающийся: _____ Шклярова Ксения Алексеевна

Руководитель: _____ Курочкин Михаил Александрович

«_____»_____20___г.

Санкт-Петербург, 2025

Содержание

Введение

Тестирование программного обеспечения – это процесс или последовательность процессов, позволяющих удостовериться в том, что программный код делает все то, для чего он предназначался, и, наоборот, не делает того, для чего не предназначался.

Ручное тестирование — это процесс тестирования, не предполагающий использования компьютера и осуществляемый непосредственно человеком. Ручное тестирование позволяет выявить ошибки, которые могут быть не замечены при автоматическом тестировании.

Тремя основными методами ручного тестирования являются инспекция кода (code inspection), сквозной просмотр (walkthrough) и тестирование удобства использования (usability testing).

В данной лабораторной работе проводилось тестирование программы методом инспекции кода.

1 Изучение методов ручного тестирования

1.1 Инспекция кода и сквозные просмотры

Инспекция и сквозной просмотр включают в себя чтение или визуальную проверку программного кода группой лиц. Оба метода предполагают выполнение определенной подготовительной работы. Завершающим этапом является обмен мнениями между участниками проверки на специальном заседании. Цель такого заседания — нахождение ошибок, но не их устранение (т.е. тестирование, а не отладка).

Инспекция и сквозной просмотр - это усовершенствованные версии процесса проверки за столом (desk-checking), который заключается в просмотре программистом разрабатываемой им программы перед тестированием. Однако они гораздо более эффективны, так как в проверке участвует не только автор программы, но и другие.

Инспекция кода — это набор процедур и методик обнаружения ошибок путем анализа (чтения) кода группой специалистов. В большинстве случаев, говоря об инспекции кода, основное внимание уделяют процедурным вопросам, формам, подлежащим заполнению, и т.п.

Сквозной просмотр кода — это процесс обнаружения ошибок в коде путём его просмотра участниками группы тестирования. Он похож на инспектирование, но использует другие процедуры и методы. Тестировщик приходит на совещание с подготовленными тестами — наборами входных и ожидаемых выходных данных для программы или модуля. Во время заседания участники «прогоняют в уме» каждый тест, обрабатывая данные вручную в соответствии с логикой программы. Состояние программы отслеживается на бумаге или доске.

1.2 Группа инспектирования кода

Обычно в состав группы входят четыре человека, один из которых играет роль координатора, аналогичную в данном контексте роли инженера ОТК. Координатор должен быть квалифицированным программистом, но не автором тестируемой программы, детальное знание которой от него не требуется. В его обязанности входит следующее:

1. Раздача материалов участникам заседания инспекционной группы и составление плана его проведения;
2. Проведение заседания;
3. Запись всех обнаруженных ошибок;
4. Последующая проверка того, что обнаруженные ошибки устранены.

Вторым участником группы является программист, а остальными — проектировщик программы (если это не сам программист) и специалист по тестированию. Последний должен не только

хорошо ориентироваться в методах тестирования ПО, но и знать наиболее распространенные типы ошибок, о чем будет говориться далее.

1.3 Порядок проведения заседаний по инспектированию кода

За несколько дней до заседания координатор раздает листинг программы и спецификацию проекта всем участникам группы. Это делается для того, чтобы к моменту проведения заседания участники успели ознакомиться с необходимыми материалами. Инспекционное заседание проводится следующим образом.

1. Программист рассказывает присутствующим о логике работы программы, подробно останавливаясь на каждой инструкции. В это время другие участники заседания должны задавать ему вопросы, которые поспособствуют выявлению возможных ошибок.

2. Участники заседания анализируют код программы, сверяясь с составленными на основе имеющегося опыта контрольными списками наиболее распространенных ошибок.

Координатор обязан направлять дискуссию в конструктивное русло и следить за тем, чтобы участники концентрировали свое внимание не на устранении ошибок, а на их обнаружении.

По окончании заседания программисту передастся список найденных ошибок. Если этот список достаточно длинный или устранение ошибок требует внесения существенных изменений в программу, координатор может принять решение о повторной инспекции кода после того, как ошибки будут исправлены. Список анализируется и после разбиения ошибок на категории дополняет имеющийся контрольный список, что позволяет повысить эффективность будущих инспекций кода.

Время и место проведения инспекции должны быть спланированы так, чтобы никакие внешние факторы не могли мешать работе заседания. Оптимальная длительность таких заседаний составляет от полутора до двух часов.

1.4 Категории ошибок при инспекции кода

Существуют основные категории ошибок, которые являются при инспекции программного кода:

1. Ошибки обращения к данным
2. Ошибки описания данных
3. Ошибки вычислений
4. Ошибки сравнения
5. Ошибки в управляющей логике

6. Ошибки программных интерфейсов

7. Ошибки ввода-вывода

В соответсвии с каждой категорией существует список вопросов, рекомендованных для анализирования программного кода.

2 Постановка задач

Цель лабораторной работы: проведение процедуры инспекции кода в роли разработчика.

Задачи данной лабораторной работы:

1. Изучить методы ручного тестирования;

2. Провести инспекцию кода в роли разработчика;

3. Скорректировать код программы с учетом рекомендаций инспектора.

Для инспекции кода была выбрана программа, представляющая собой калькулятор «большой» конечной арифметики $< \mathbb{Z}_i; +, * >$ (8 разрядов) для трех действий $(+, -, *)$ на основе «малой» конечной арифметики, где задано правило «+1» и выполняются свойства коммутативности $(+, *)$, ассоциативности $(+, *)$, дистрибутивности $*$ относительно $+$, заданы аддитивная единица «a» и мультипликативная единица «b», а также выполняется свойство: для $(\forall x) x * a = a$. Правило «+1» приведено в таблице:

	a	b	c	d	e	f	g	h
+1	b	c	g	h	a	e	d	f

Реализация включает в себя следующие функции: сумма, разность и произведение. Так же реализовано пользовательское меню, позволяющее выбирать арифметическую операцию по букве: 'a' для сложения, 'b' для разности первого и второго числа, 'c' для разности второго и первого числа, 'd' для произведения.

Входные данные: строка, являющаяся первым числом, строка, являющаяся вторым числом, символ для выбора ариыметического действия.

Выходные данные: результат выполненной операции.

Ограничения: задать число и выбрать арифметическое действие из пользовательского меню можно следующими символами: «a», «b», «c», «d», «e», «f», «g», «h», все символы должны быть нижнего регистра; максимальное число разрядов заданного числа - 8.

3 Исходный код программы

```
vector<string> my_vect{"a", "b", "c", "g", "d", "h", "f", "e"};
vector<char> my_vectt{'a', 'b', 'c', 'g', 'd', 'h', 'f', 'e'};

bool isMyLetter(string const& str){
    return !str.empty() && str.find_first_not_of("abcdefgh") == string::npos;
}

char plus_razr(char c, char s){
    int a = 0;
    for (int i = 0; i < 8; i++){
        if (s == my_vectt[i]){
            a = i;
        }}
    while (a != 0){
        c = plus_1(c);
        a--;
    }
    return c;
}

string sum(string c, string s){
    vector<char> my_v(16, 'a');
    string ss;

    if (c.size() == s.size()){
        for (int i = 0; i < max(c.size(), s.size()); i++){
            char b = my_v[i];
            my_v[i] = plus_razr(c[c.size()-1-i], my_v[i]);
            my_v[i] = plus_razr(s[s.size()-1-i], my_v[i]);
            if (razr3(c[c.size()-1-i], s[s.size()-1-i], b)){
                my_v[i+1] = plus_razr(my_vectt[1], my_vectt[0]);
            }}}
    if (c.size() > s.size()){
        for (int i = 0; i < s.size(); i++){
            char b = my_v[i];
            my_v[i] = plus_razr(c[c.size()-1-i], my_v[i]);
            my_v[i] = plus_razr(s[s.size()-1-i], my_v[i]);
            if (razr3(c[c.size()-1-i], s[s.size()-1-i], b)){
                my_v[i+1] = plus_razr(my_vectt[1], my_vectt[0]);
            }}
        for (int i = s.size(); i < c.size(); i++){
            char b = my_v[i];
            my_v[i] = plus_razr(c[c.size()-1-i], my_v[i]);
            if (razr(c[c.size()-1-i], b)){
                my_v[i+1] = plus_razr(my_vectt[1], my_vectt[0]);
            }}}
    if (c.size() < s.size()){
        for (int i = 0; i < c.size(); i++){
            char b = my_v[i];
```

```
            my_v[i] = plus_razr(c[c.size()-1-i], my_v[i]);
            my_v[i] = plus_razr(s[s.size()-1-i], my_v[i]);
            if (razr3(c[c.size()-1-i], s[s.size()-1-i], b)){
                my_v[i+1] = plus_razr(my_vectt[1], my_vectt[0]);
            }}
        for (int i = c.size(); i < s.size(); i++){
            char b = my_v[i];
            my_v[i] = plus_razr(s[s.size()-1-i], my_v[i]);
            if (razr(s[s.size()-1-i], b)){
                my_v[i+1] = plus_razr(my_vectt[1], my_vectt[0]);
            }}}
    int f = 9;
    for (int i = 0; i < my_v.size(); i++){
        if (my_v[i] != 'a'){
            f = i;
        }}
    for (int i = f; i >= 0; i--){
        ss.push_back(my_v[i]);
    }
    return ss;
}

string minus__(string c, string s){
    vector<char> vect(c.size(), 'a');
    int a = 0, b = 0;
    for (int i = 0; i < s.size(); i++){
        for (int j = 0; j < 8; j++){
            if (c[c.size()-1-i] == my_vectt[j]){
                a = j;
            }
            if (s[s.size()-1-i] == my_vectt[j]){
                b = j;
            }}
        if (a >= b){
            vect[i] = minus_razr(c[c.size()-1-i], s[s.size()-1-i]);
        }
        else{
            char d = my_vectt[7];
            vect[i] = plus_1(plus_razr(minus_razr(d, s[s.size()-1-i]),
            c[c.size()-1-i]));
            for (int j = c.size()-i-2; j >= 0; j--){
                if (c[j] != my_vectt[0]){
                    c[j] = minus_1(c[j]);
                    break;
                }
                else{
                    c[j] = my_vectt[7];
                }}}}
    for (int i = s.size(); i < c.size(); i++){
        vect[i] = c[c.size()-1-i];
    }
```

```
    string ss;
    int f = 0;
    for (int i = 0; i < vect.size(); i++){
        if (vect[i] != 'a'){
            f = i;
        }}
    for (int i = f; i >= 0; i--){
        ss.push_back(vect[i]);
    }
    return ss;
}

string umnoj(string c, string s){
    if (c == my_vect[0] || s == my_vect[0]){
        return my_vect[0];
    }
    else{
        vector<string> vect;
        for (int i = 0; i < s.size(); i++){
            vector<char> v(16, 'a');
            for (int j = 0; j < c.size(); j++){
                int t = 0;
                for (int h = 0; h < 8; h++){
                    if (v[j+i] == my_vectt[h]){
                        t = h;
                        break;
                    }}
                string ff = sum(umnoj_razr(s[s.size()-1-i],
                c[c.size()-1-j]), my_vect[t]);
                if (ff.size() == 1){
                    v[j+i] = ff[0];
                }
                else{
                    v[j+i] = ff[1];
                    v[j+1+i] = ff[0];
                }}
            string ss;
            for (int j = v.size()-1; j >= 0; j--){
                ss.push_back(v[j]);
            }
            vect.push_back(ss);
        }
        string str = "a";
        for (int i = 0; i < vect.size(); i++){
            str = sum(str, vect[i]);
        }
        return str;
    }}

void menu(){
    cout << "Выберете значение A" << endl;
```

```
152     A = get_numb();
153     cout << "Выберете значение B" << endl;
154     B = get_numb();
155     cout << "A = " << A << endl << "B = " << B << endl;
156     cout << "Действие над множетсвами" << endl;
157     printf(" a. A + B \n b. A - B \n c. B - A \n d. A * B");
158     for (string z; ; getline(cin, z)){
159         if (z.length() == 1){
160             switch(*z.c_str()){
161                 case 'a':
162                     cout << "A + B = ";
163                     if (sum(A,B).size() > 8){
164                         cout << "переполнение" << endl;
165                     }
166                     else{
167                         cout << sum(A, B) << endl;}
168                     break;
169                 case 'b':
170                     cout << "A - B = ";
171                     if (minus_(A,B).size() > 8){
172                         cout << "переполнение" << endl;
173                     }
174                     else{
175                         cout << minus_(A, B) << endl;}
176                     break;
177                 case 'c':
178                     cout << "B - A = ";
179                     if (minus_(B,A).size() > 8){
180                         cout << "переполнение" << endl;
181                     }
182                     else{
183                         cout << minus_(B, A) << endl;}
184                     }
185                     break;
186                 case 'd':
187                     cout << "A * B = ";
188                     if (umnoj(A,B).size() > 8){
189                         cout << "переполнение" << endl;
190                     }
191                     else{
192                         cout << umnoj(A, B) << endl;}
193                     break;
194                 default:
195                     cout << "Выберите интересующее вас" << endl;
196                     break;
197             }}}
198 }
```

4 Описание проведенной инспекции кода

В начале заседания программистом была описана логика программы. В процессе инспекции были заданы вопросы с целью выявления ошибок. Далее представлены заданные вопросы и ответы на них.

Участники заседания:

– Программист, разработчик программы – Шклярова Ксения;

– Специалист по тестированию – Черепанов Михаил;

– Секретарь – Григорьев Пётр.

4.1 Протокол заседания

1. Используются ли переменные с неинициализированными значениями

   - Да

2. Выходят ли индексы за границы массива

   - Нет

3. Используются ли нецелочисленные индексы

   - Нет

4. Есть ли “висячие” ссылки?

   - Нет

5. Корректны ли атрибуты всех псевдонимов

   - Да

6. Согласуются ли атрибуты структуры данных и физической записи

   - Да

7. Согласуются ли описания структуры данных в разных процедурах?

   - Да

8. Существуют ли ошибки завышения или занижения значения индекса на единицу при индексации массивов?

   - Нет

9. Участвуют ли в вычислениях неарифметические переменные

   - Нет

10. Выполняются ли вычисления с сипользование данных разного типа

    - Нет

11. Выполняются ли вычисления с переменными разной длины

    - Да

12. Присваиваются ли переменным значения типа данных большего размера?

    - Нет

13. Возможны ли переполнение или потеря точности в процессе промежуточных вычислений?

    - Нет

14. Возможно ли деление на нуль?

    - Нет

15. Не выходят ли значения переменных за логически обоснованные пределы?

    - Да

16. Все ли переменные объявлены

    - Да

17. Правильно ли инициализированы массивы и строки?

    - Да

18. Правильно ли определены размеры, типы и классы памяти?

    - Да

19. Корректно ли используются булевы выражения?

    - Да

20. Есть ли попытки сравнения переменных разного типа?

    - Нет

21. Есть ли попытки сравнения несопоставимых переменных?

    - Нет

22. Корректно ли используются операторы сравнения?

    - Да

23. Понятны ли приоритеты операций?

    - Да

24. Понятен ли способ разбора булевых выражений компилятором?

    - Да

25. Может ли значение индекса в переключателе превысить число переходов?

    - Нет

26. Будет ли завершен каждый цикл?

    - Да

27. Будет ли завершена программа?

    - Да

28. Есть ли цикл, который может не выполниться из-за входных условий?

    - Нет

29. Корректны ли возможные погружения в цикл?

    - Да

30. Имеются ли ошибки диапазона, приводящие к отклонению количества итераций от нужного значения?

    - Нет

31. Существуют ли точки ветвления, в которых не учтены все возможные условия?

    - Нет

32. Совпадает ли число формальных параметров с числом аргументов?

    - Да

33. Совпадают ли атрибуты параметров и аргументов?

    - Да

34. Совпадает ли число аргументов, передаваемых вызываемым модулям, с числом ожидаемых параметров?

    - Да

35. Правильно ли заданы количество, атрибуты и порядок следования аргументов при вызове встроенных функций?

    - Да

36. Делаются ли в подпрограмме попытки изменить входные аргументы?

    - Нет

37. Передаются ли константы в качестве аргументов?

    - Нет

38. Присутствуют ли в выходной информации орфографические или грамматические ошибки?

    - Нет

39. Обрабатываются ли ошибки ввода вывода?

    - Нет

40. Осуществляется ли проверка корректности входных данных?

    - Да

41. Нет ли пропущенных функций?

    - Нет

42. Совпадает ли список атрибутов с тем, который следовало ожидать?

    - Да

4.2 Полученные рекомендации

1. Переименовать переменные и функции для улучшения читаемости.

Переменные, такие как my_vect и my_vectt, лучше переименовать для лучшего понимания их назначения. Например, my_vect можно назвать letters, а my_vectt — letter_values.

2. В функции умножения перенести проверку нулевых значений из середины программы в начало, для проверки до выполнения операций.

3. Не смотря на наличие проверки корректности входных данных, лучше добавить обработку ошибок или исключений для случаев, когда строки не соответствуют ожидаемым значениям.

4. Оптимизировать циклы в sum, так как в коде есть избыточные циклы и проверки.

5. Добавить комментарии для большего понимания кода.

4.3 Исправленный код

```
// Переименование для улучшения читаемости
vector<string> letters{"a", "b", "c", "g", "d", "h", "f", "e"};
vector<char> letter_values{'a', 'b', 'c', 'g', 'd', 'h', 'f', 'e'};

// Функция проверки строки на допустимые символы
bool isValidLetter(const string& str) {
    if (!str.empty() && str.find_first_not_of("abcdefgh") == string::npos){
        return 1;
    }
    else{
        cout << "Строка содержит недопустимые символы. Допустимы только символы: a, b, c, g, d
            , h, f, e." << endl;
        return 0;
    }
}

// Функция для прибавления разряда
char increment(char c, char s) {
    int a = 0;
    for (int i = 0; i < 8; i++) {
        if (s == letter_values[i]) {
            a = i;
            break;
        }}
    while (a != 0) {
        c = plus_1(c);
        a--;
    }
    return c;
}

//Функция для прибасления 1
char plus_1(char c){
    char s = 0;
    for (int i = 0; i < 8; i++){
        if (c == letter_values[i]){
            if (c == letter_values[7]){
                s = letter_values[0];
                break;
            }
            s = letter_values[i+1];
            break;
        }
    }
    return s;
}

// Функция для выполнения сложения
string add(string c, string s) {
```

```
    vector<char> result(16, 'a');
    string ss;
    // Обработка случая, когда строки имеют одинаковую длину
    int maxLength = max(c.size(), s.size());
    for (int i = 0; i < maxLength; i++) {
        char carry = result[i];
        result[i] = increment(c[c.size() - 1 - i], result[i]);
        result[i] = increment(s[s.size() - 1 - i], result[i]);
        if (checkCarry(c[c.size() - 1 - i], s[s.size() - 1 - i], carry)) {
            result[i + 1] = increment(letter_values[1], letter_values[0]);
        }}
    // Преобразуем результат в строку
    int f = 9;
    for (int i = 0; i < result.size(); i++) {
        if (result[i] != 'a') {
            f = i;
        }}
    for (int i = f; i >= 0; i--) {
        ss.push_back(result[i]);
    }
    return ss;
}

//Функция для вычитания 1
char decrement(char c){
    char s = 0;
    for (int i = 1; i < 8; i++){
        if (c == letter_values[i]){
            s = letter_values[i-1];
            break;
        }
    }
    return s;
}

// Функция для выполнения вычитания
string subtract(string c, string s) {
    vector<char> result(c.size(), 'a');
    int a = 0, b = 0;
    for (int i = 0; i < s.size(); i++) {
        for (int j = 0; j < 8; j++) {
            if (c[c.size() - 1 - i] == letter_values[j]) {
                a = j;
            }
            if (s[s.size() - 1 - i] == letter_values[j]) {
                b = j;
            }}
        if (a >= b) {
            result[i] = subtractDigits(c[c.size() - 1 - i], s[s.size() - 1 - i]);
        }
        else {
```

```
            char borrow = letter_values[7]; // Перенос разряда
            result[i] = increment(increment(subtractDigits(borrow, s[s.size() - 1 - i]), c[c.
                size() - 1 - i]), letter_values[0]);
            for (int j = c.size() - i - 2; j >= 0; j--) {
                if (c[j] != letter_values[0]) {
                    c[j] = decrement(c[j]);
                    break;
                } else {
                    c[j] = letter_values[7];
                }}}}
    // Обработка оставшихся разрядов
    for (int i = s.size(); i < c.size(); i++) {
        result[i] = c[c.size() - 1 - i];
    }
    string ss;
    int f = 0;
    for (int i = 0; i < result.size(); i++) {
        if (result[i] != 'a') {
            f = i;
            }}
    for (int i = f; i >= 0; i--) {
        ss.push_back(result[i]);
    }
    return ss;
}

// Функция для выполнения умножения
string multiply(string c, string s) {
    if (c == letters[0] || s == letters[0]) {
        return letters[0];
    } else {
        vector<string> resultStrings;
        for (int i = 0; i < s.size(); i++) {
            vector<char> result(16, 'a');
            for (int j = 0; j < c.size(); j++) {
                int t = 0;
                for (int h = 0; h < 8; h++) {
                    if (result[j + i] == letter_values[h]) {
                        t = h;
                        break;
                    }}
                string product = add(multiplyDigits(s[s.size() - 1 - i], c[c.size() - 1 - j]),
                     letter_values[t]);
                if (product.size() == 1) {
                    result[j + i] = product[0];
                } else {
                    result[j + i] = product[1];
                    result[j + 1 + i] = product[0];
                }}
            string resultStr;
            for (int j = result.size() - 1; j >= 0; j--) {
```

```
                resultStr.push_back(result[j]);
            }
            resultStrings.push_back(resultStr);
        }
        string finalResult = "a";
        for (int i = 0; i < resultStrings.size(); i++) {
            finalResult = add(finalResult, resultStrings[i]);
        }
        return finalResult;
    }}

// Функция меню
void menu() {
    cout << "Выберите значение |||first_number|||: ";
    string first_number = get_numb();
    cout << "Выберите значение |||second_number|||: ";
    string second_number = get_numb();
    cout << "first_number = " << first_number << endl << "second_number = " << second_number
        << endl;
    cout << "Действие над множествами:" << endl;
    printf("a. first_number + second_number \n b. first_number - second_number \n c.
        second_number - first_number \n d. first_number * second_number\n");

    while (true) {
        string z;
        getline(cin, z);
        if (z.length() == 1) {
            switch (z[0]) {
                case 'a':
                    cout << "|||first_number + second_number = |||";
                    if (add(first_number, second_number).size() > 8) {
                        cout << "переполнение" << endl;
                    } else {
                        cout << add(first_number, second_number) << endl;
                    }
                    break;
                case 'b':
                    cout << "first_number - second_number = ";
                    if (subtract(first_number, second_number).size() > 8) {
                        cout << "переполнение" << endl;
                    } else {
                        cout << subtract(first_number, second_number) << endl;
                    }
                    break;
                case 'c':
                    cout << "|||second_number - first_number = |||";
                    if (subtract(second_number, first_number).size() > 8) {
                        cout << "переполнение" << endl;
                    } else {
                        cout << subtract(second_number, first_number) << endl;
                    }
```

```
                    break;
                case 'd':
                    cout << "|||first_number * second_number = |||";
                    if (multiply(first_number, second_number).size() > 8) {
                        cout << "переполнение" << endl;
                    } else {
                        cout << multiply(first_number, second_number) << endl;
                    }
                    break;
                default:
                    cout << "Выберите интересующее вас действие!" << endl;
                    break;
            }}}
}
```

Заключение

В ходе выполнения лабораторной работы мной были изучены методы ручного тестирования, а так же был получен практический опыт выявления ошибок в программном коде путем его внимательного прочтения и анализа, а также при ответах на вопросы, задаваемые специалистом по тестированию.

Была выполнена инспекция кода, направленная на выявление дефектов и улучшение качества программы. В результате инспекции был обнаружен ряд ошибок и недочетов, а также сформулированы рекомендации для программиста по их устранению. После получения рекомендаций были внесены необходимые исправления в код.

Проведенная инспекция выявила следующие недочеты: некорректный выбором имен переменных и функций, что затрудняло понимание кода, отсутствие комментариев, избыточные циклы и проверки, отсутствие обработки входных значений. Устранение этих недочетов позволило повысить не только понимание, но и производительность программы.