Министерство образования и науки
Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и кибербезопасности
Высшая школа технологий искусственного интеллекта
Направление: 02.03.01 Математика и компьютерные науки

Отчет по дисциплине

«Методы тестирования программного обеспечения»

«Инспекция кода»

Выполнил: студент, гр. 5130201/20102
_____ Салимли Айзек

Преподаватель: к.т.н.
_____ 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.

Хороший тест - тот, который обеспечивает высокую вероятность обнаружения еще не вы-явленных ошибок. Успешное тестирование подразумевает детальное описание ожидаемых значений не только на входе, но и на выходе программы. Одним из методов тестирования является ручное тестирование.

Ручное тестирование (human testing) - процесс тестирования, не предполагающий использования компьютера и осуществляемый непосредственно человеком. Методы ручного тестирования должны применяться к написанному исходному коду до того, как начнется его тестирование на компьютере. Применение неформальных методов не только не препятствует успешному тестированию, но и благотворно сказывается на производительности и надежности этого процесса. Одним из методов ручного тестирования является метод - Инспекции кода.

Инспекция кода - это набор процедур и методик обнаружения ошибок путём анализа (чтение) группы специалистов.

В ходе работы была проведена инспекция кода. В данном отчёте приведено описание метода инспекции кода с использованием списка контрольных вопросов, порядок проведения и результат данного метода ручного тестирования.

1 Постановка задачи

1. Изучить метод ручного тестирования - инспекция кода
2. Провести инспекцию кода в роли разработчика-программиста
3. Проанализировать результат инспекции кода
4. Внести изменения в программный код

2 Инспекция кода

Инспекция кода - это набор процедур и методик обнаружения ошибок путём анализа (чтение) кода группа специалистов. В большинстве случаев, говоря об инспекции кода, основное внимание уделяют процедурном вопросам, формам, подлежащим заполнению, и т.п.

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

- Раздача материалов участникам заседания инспекционной группы и составление плана его проведения
- Запись всех обнаруженных ошибок
- Последующая проверка того что обнаружены 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 распространённый типа ошибок, о чем будет говорится далее.

3 Порядок проведения инспекционного заседания

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. Инспекционные заседание проводится следующим образом:

1. Программист рассказывает присутствующим о логике работы программы, подробно останавливаюсь на каждой инструкции. В это время другие участники заседания должны задавать ему вопросы, которые по способствуют выявлению возможных ошибок. Вполне вероятно, что по ходу дела часть ошибок обнаружат сам программист, а не остальные участники группы.
2. Участники заседания анализирует код программы, сверяя составленными на основе имеющегося опыта контрольными списками наиболее распространённых ошибок (подобные список обсуждается в следующем разделе).

Координатор обязан направлять дискуссию в конструктивное русло и следить затем, чтобы участники концентрировали свое внимание не на устранение ошибок, а на их обнаружение (соответствующие исправления программист внесёт самостоятельно после заседания).

По окончании заседания программисту передается список найденных ошибок. Оптимальное длительность таких заседаний составляет от полутора до двух часов.

4 Контрольные списки возможных ошибок для инспекции кода

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

- Ошибки обращения к данным
- Ошибки описания данных
- Ошибки вычислений
- Ошибки сравнения
- Ошибки в управляющей логике
- Ошибки ввода вывода и др.

В соответствии с каждой категории существующих списков вопросов, рекомендованных для анализирования программного кода.

5 Описание программы

Название: базовая языковая модель на основе N-граммного словаря на языке программирования Haskell.

Дано:

- Текстовый файл с расширением .txt, содержащий произвольный текст на естественном языке.
- Тип входных данных: String (строки), Int (количество ключа-слов).

Требуется:

- Программно обработать текст, разбив его на отдельные предложения и нормализовать текст (приведение к нижнему регистру, удаление знаков препинания).
- На основе обработанного текста создать словарь словосочетаний длиной от одного заданного числа слов (n). Словари содержит возможные варианты продолжения слов или словосочетаний.
- Реализовать диалоговую систему с двумя собеседниками, где каждая следующее высказывания генерируется на основе последних слов предыдущего высказывания с помощью соответствующего словаря.
- Генерировать реплики случайным образом из подходящих по ключу вариантов предложения.

Выход:

- Сгенерированный фразой диалога между двумя собеседниками выводятся в терминал интегрированный среды разработки.
- Тип выходных данных: String (строки).

5.1 Спецификация

На вход программа получает имя текстового файла с расширением .txt, содержащего произвольный текст на естественном языке (строчные и прописные буквы, знаки препинания и пробельные символы).

Текст файла будет использоваться для генерации словаря возможных продолжений слов и словосочетаний длиной от одного до заданного числа n.

После успешного считывания и обработки текста из файла, программа генерирует диалог между двумя участниками, последовательно создавая реплики на основе последних слов предыдущей реплики и соответствующего словаря.

Каждая реплика генерируется случайным образом из имеющихся вариантов продолжения, определённых по ключу (последним словам текущей фразы). Диалог продолжается указанное пользователем количество шагов.

В процессе генерации диалога возможны следующие исключительные ситуации:
- Если дальнейшее продолжение диалога невозможно (отсутствуют подходящие ключи или продолжения в словаре), программа выводит сообщение об ошибке вида «Ошибкочка!», «Завершен!» или «Ключ N найден но продолжение отсутствует» и завершает выполнение.
- Если ключ для следующей реплики отсутствует во всех словарях, программа выводит сообщение «Ключ не найден !!!» и завершает выполнение.

Тип входных данных:

- Имя файла (с указанием расширения), содержащего текстовые данные для обработки.
- Натуральное число n, задающее максимальную длину ключевых слов и их продолжений в словаре.
- Начальные слова (реплика), с которых начинается диалог.
- Число шагов (реплик), которые необходимо сгенерировать.

Тип входных данных: String, Int.

Тип выходных данных: вывод на экран (String).

Примечание*: программа не генерирует исключений, а информирует об ошибках через сообщения и завершает работу штатно.

Таб. 1: Спецификация

Входные данные	Выходные данные	Реакция программы
Текстовый файл - Check1.txt, содрежащий текст: We will see them, sure, tomorrow! Количество ключ-слов словаря - 3	1: we will see them sure tommorow 2: we : will , see , them , tomorrow will : we , see , them , tomorrow see : we , will , them , tomorrow them : we , will , see , tomorrow tomorrow : we , will , see , them see : them , tomorrow , them , will we will : see , them , tomorrow, sure sure we : will , them , see , tomorrow … tomorrow then sure see we will : [ ] 3: sure : we will see 4: A1- will we see A2- tomorrow sure	Процесс завершен с возвратом монады Just (Right).
Текстовый файл - Check2.txt, содрежащий текст: #@#$@#!!!$($#_~~!!!	Ошибка: Файл пуст или не содержит естественного языка!	Процесс завершен с возвратом монады Nothing (Left).
Пустой текстовый файл - Check3.txt	Ошибка: Файл пуст или не содержит естественного языка!	Процесс завершен с возвратом монады Nothing (Left).
Файл другого текстового формата - Chek4.rtx	Ошибка формата! Файл должен быть .txt	Процесс завершен с возвратом монады Nothing (Left).
Текстовый файл - Check5.txt, содрежащий текст: 33345355262642	Ошибка: Файл пуст или не содержит естественного языка!	Процесс завершен с возвратом монады Nothing (Left).
Файл не текстового формата - Check6.jpg, содержащий изоброжение.	Ошибка формата! Файл должен быть .txt	Процесс завершен с возвратом монады Nothing (Left).
Текстовый файл - Check7.txt, содрежащий текст на кирилице: Привет друг!	1: привет друг 2: привет: друг привет друг: [ ] друг: привет друг привет: [ ] 3: друг привет 4: A1- привет друг A2 - друг привет	Процесс завершен с возвратом монады Just (Right).

5.2 Математическое описание

В нашем случае реализована Базовая N-граммная модель, без введения правил грамматик и вероятностей.

N-граммные языковые модели используют предположение Маркова, которые утверждает что в контексте языкового моделирования вероятность следующего слова в последовательности зависит только от предыдущих слов. Следовательно, приближение вероятности слова, учитывая его контекст в n-граммной модели, можно формализовать следующим образом:

$$P(w_t \mid w_{t-1}) \approx P(w_t \mid w_{t-N+1:t-1})$$

где t - это количество слов во всей последовательности, а N - размер контекста (униграмма (1), биграмма (2), … , N-грамма (N)).

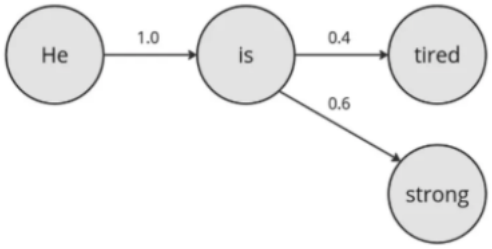

Рис. 1: Цепь Маркова перехода слов

Для такого представления вероятность можно определить такой формулой:

$$P(w_t \mid w_{t-N+1}^{t-1}) \approx \frac{C(w_{t-N+1}^{t-1} w_t)}{C(w_{t-N+1}^{t-1})}$$

Вероятность следующего слова, учитывая N предыдущих: числитель – сколько раз результирующая последовательность встречается в данных, знаменатель – сколько раз последовательность предыдущих слов встречается в данных.

N-граммная модель с использованием реальных вероятностей переходов слов реализована в библиотеках LLM.

5.3 Метод решения

Код предоставленный на инспекцию решает задачу следующим алгоритмом:

Предварительная обработка текста:

Исходный текст разбивается на отдельные предложения.

Выполняется нормализация текста:

- перевод всех символов в нижний регистр;
- удаление знаков препинания и других символов, кроме букв и пробелов.

Формирование словаря N-грамм:

- Текст разбивается на последовательности из N слов (ключ) и следующие за ними слова (значения).

Создается словарь, где:

- ключом является последовательность из 1 до N слов;
- значением — список возможных продолжений (слов или фраз), следующих за данным ключом в исходном тексте.

Генерация фразы:

Фраза генерируется по принципу случайного выбора следующего слова или слов из множества возможных продолжений, основываясь на текущем контексте (последние N-1 слов текущей фразы).

Генерация продолжается до тех пор, пока:

- достигается максимальная длина фразы, заданная заранее;
- отсутствует подходящее продолжение в словаре (происходит прерывание генерации с выводом соответствующего сообщения об ошибке).

Диалоговая логика:

- Процесс генерации реплик реализуется в виде диалога между двумя агентами.
- Реплики генерируются поочерёдно, каждый раз используя словарь, привязанный к текущему агенту.
- Реплики второго участника строятся на основе последних слов реплики первого, и наоборот.
- Если ключ не найден в словаре, алгоритм постепенно укорачивает ключ (убирая слова слева направо), пока не найдёт подходящий ключ либо полностью не исчерпает слова.

6 Исходный код программы

Программа собрана в stack-проект. Управляющая логика программы написана в файле Src/Lib.hs:

```
{-# LANGUAGE LambdaCase #-}
module Lib (prodlojaNorm, nSlovarchik, saveSlovarchik, slovoKomputera, dialog) where
import Data.Char (isLetter, toLower)
import Data.List (nub, intercalate)
import Data.List.Split (splitWhen)
import Data.Map.Strict (Map)
import qualified Data.Map.Strict as Map
import System.Random (randomRIO)

prodlojaNorm :: String -> Maybe [[String]]
prodlojaNorm text
    | null text = Nothing
    | otherwise = Just $ filter (not . null) $ map (words . normPredloja) $ predlojaS
text
  where
    razdelit :: String
    razdelit = ".!?;:()"
    predlojaS :: String -> [String]
    predlojaS t = splitWhen (`elem` razdelit) t
    normPredloja :: String -> String
    normPredloja = map toLower . filter (\c -> isLetter c || c == ' ')

nSlovarchik :: Int -> [[String]] -> Map String [String]
nSlovarchik n predlojaS =
  Map.filter (not . null) $
    Map.fromListWith (\new old -> nub (old ++ new)) $
      [ (unwords key, [value])
        | predlojNSlovar <- predlojaS
        , (key, value) <- slovarVPredloj n predlojNSlovar
      ]
  where
    slovarVPredloj :: Int -> [String] -> [([String], String)]
    slovarVPredloj n words =
      [ (key, unwords value)
        | i <- [0 .. length words - 1]
        , k <- [1 .. min (n - 1) (length words - i)]
        , let key = take k (drop i words)
        , let rest = drop (i + k) words
        , not (null rest)
        , l <- [1 .. min (n - k) (length rest)]
        , let value = take l rest
        , not (null value)
      ]++
      [ (key, "")
        | i <- [max 0 (length words - (n - 1)) .. length words - 1], i >= 0
        , k <- [1 .. min (n - 1) (length words - i)]
        , let key = take k (drop i words)
        , length key == k
        , drop (i + k) words == []
      ]
saveSlovarchik :: FilePath -> Map String [String] -> IO ()
saveSlovarchik filePath dict =
    let vhod = Map.toAscList dict
        filterVhod = filter (not . null . snd) vhod
        formattedvhod = map formatNorm filterVhod
    in writeFile filePath (unlines formattedvhod)
  where
    formatNorm (key, values) =
      let values' = nub values
```

```
          nePusto = if all null values' then [] else filter (not . null) values'
          valuesStr = intercalate ", " (map show nePusto)
      in key ++ " : [" ++ valuesStr ++ "]"

slovoKomputera :: Map String [String] -> [String] -> IO (Either String [String])
slovoKomputera dict keyWords = generate (unwords keyWords) [] maxLength
  where
    maxLength = 15
    minLength = 2
    generate _ acc 0ы
        | length acc >= minLength = return (Right (reverse acc))
        | otherwise = return (Left "Ошибкочка!")
    generate curKey acc n =
        case Map.lookup curKey dict of
            Nothing
                | length acc >= minLength -> return (Right (reverse acc))
                | otherwise -> return (Left $ "Завершен!")
            Just [] -> return (Left $ "Ключ '" ++ curKey ++ "' найден но продолжение
отсутствует(")
            Just nextOptions ->
                randomRIO (0, length nextOptions - 1) >>= \idx ->
                let nextValue = nextOptions !! idx
                in if null nextValue
                   then return (Right (reverse acc))
                   else let continuation = words nextValue
                        in generate (unwords $ drop 1 (words curKey ++ continuation))
(continuation ++ acc) (n - 1)

keyPoUbiv :: [String] -> Map String [String] -> Maybe String
keyPoUbiv [] _ = Nothing
keyPoUbiv (x:xs) dict
    | Map.member x dict = Just x
    | otherwise = keyPoUbiv xs dict

dialog :: Map String [String] -> Map String [String] -> [String] -> Int -> IO ()
dialog dict1 dict2 initialPhrase steps =
    let logDialog prevPhrase d1 d2 remainingSteps sobesNumber
            | remainingSteps == 0 = putStrLn "Диалог завершён"
            | otherwise =
                let currentsobes = if sobesNumber `mod` 2 == 1 then 1 else 2
                    dict = if currentsobes == 1 then d1 else d2
                    otherDict = if currentsobes == 1 then d2 else d1
                    sobesLabel = "Чел " ++ show currentsobes ++ ": "
                in slovoKomputera dict prevPhrase >>= \case
                    Left err -> putStrLn (sobesLabel ++ err) >> putStrLn "Диалог
завершён"
                    Right [] -> putStrLn (sobesLabel ++ "Ошибочка генерации!(") >>
putStrLn "Диалог завершён"
                    Right generatedPhrase ->
                        let phraseWithInitial = unwords prevPhrase ++ " " ++ unwords
generatedPhrase
                            lastWords = reverse (words phraseWithInitial)
                        in putStrLn (sobesLabel ++ "(Ключ: " ++ unwords prevPhrase ++
") " ++ phraseWithInitial) >>
                           case keyPoUbiv lastWords otherDict of
                               Nothing -> putStrLn "Ключ не найден !!!"
                               Just nextKey -> let nextPhrase = words nextKey
                                               in logDialog nextPhrase d1 d2
(remainingSteps - 1) (sobesNumber + 1)
    in logDialog initialPhrase dict1 dict2 steps 1
```

7 Инспекционное заседание

В заседании участвовали 3 человека:

- Специалист по тестированию - Черепанов Михаил
- Секретарь - Григорьев Петр
- Программист, разработчик программы - Салимли Айзек

В начале заседания участникам заседания была описана программа, ее назначение и логика работы, входные и выходные параметры.

В процессе инспекции кода задавались вопросы с целью выявить ошибки. По итогу заседания был составлен протокол заседания, который включал в себя заданные вопросы, ответы на них, а также рекомендации по устранению ошибок.

7.1 Итоги проведения заседания

В роли программиста-разработчика выступал автор отчета, Салимли Айзек. На первом этапе инспекции кода при обсуждении логики программы, мною, как разработчиком программы, был получен опыт обнаружения ошибок на этапе чтения программного кода. В дальнейшем при разработке проектов, я пользуюсь данным методом для обнаружения недочетов в программе.

Также мною были получены важные рекомендации в написании программного кода от председателя заседания.

7.2 Протокол заседания

Блок 1 - Обработка данных

Вопрос 1: Используются ли некорректные типы или структуры данных?

Ответ: ***Нет***

Вопрос 2: Используются ли некорректные типы или структуры данных?

Ответ: ***Нет***

Блок 4 - Сравнение

Вопрос 3: Сравниваются ли данные разных типов без явного преобразования?

Ответ: ***Нет***

Вопрос 4: Используется ли некорректное сравнение вещественных чисел?

Ответ: ***Нет***

Вопрос 5: Проверяется ли возможность выхода за границы диапазона при сравнении?

Ответ: ***Да***

Вопрос 6: Реализована ли проверка на сравнение с погрешностью?

Ответ: ***Нет***

Вопрос 7: Реализована ли проверка на равенство ссылок вместо значений?

Ответ: ***Да***

Вопрос 8: Учтена ли возможность неполного совпадения (частичное совпадение)?

Ответ: ***Да***

Вопрос 9: Учтена ли проверка на циклические ссылки?

Ответ: ***Да***

Блок 5 - Передача управления

Вопрос 10: Учтена ли обработка неиспользуемых ветвей кода?

Ответ: ***Да***

Вопрос 11: Реализована ли обработка альтернативных путей выполнения?

Ответ: ***Да***

*Вопрос 12: Проверяется ли выход за пределы области видимости?**

Ответ: ***Нет***

Вопрос 13: Проверяется ли зацикливание при передаче управления?

Ответ: ***Да***

Вопрос 14: Проверяется ли недостижимый код?

Ответ: ***Да***

Вопрос 15: Реализованы ли корректные условия выхода из цикла или рекурсии?

Ответ: ***Да***

Блок 6 - Другие проверки

Вопрос 16: Реализована ли обработка исключительных ситуаций?

Ответ: ***Да***

Вопрос 17: Выполняется ли проверка успешности операций ввода-вывода?

Ответ: ***Да***

*Вопрос 18: Проверяется ли корректность форматирования входных или выходных данных?**

Ответ: ***Нет***

Вопрос 19: Выполняется ли проверка утечек памяти или ресурсов?

Ответ: ***Да***

Вопрос 20: Будет ли завершена программа?

Ответ: ***Да***

*Вопрос 21: Замечены ли орфографические или грамматические ошибки в тексте, выводимой программой на печать или на экран?**

Ответ: ***Да***

Таким образом, были выявлены ошибки, которые не были замечены программистом-разработчиком в ходе написания кода. Все ошибки и недочеты в программном коде были аргументированы специалистом по тестированию.

В итоге, программистом-разработчиком были учтены замечания и рекомендации и внесены исправления в программный код.

8 Корректирование кода программы

В соответствии с данными рекомендациями на этапе инспекции кода были исправлены ошибки и недочёты в программе. Далее будут представлены фрагменты программы, которые были подтверждены корректировке. Сначала представлен первоначальный вариант, далее исправленный.

Исходный код:

```
generate _ acc 0
    | length acc >= minLength = return (Right (reverse acc))
    | otherwise = return (Left "Ошибкочка!")
```

Исправленный код: Исправлена орфографическая ошибка

```
generate _ acc 0
    | length acc >= minLength = return (Right (reverse acc))
    | otherwise = return (Left "Ошибка!")
```

Исходный код:

```
Maybe [] -> return (Left $ "Ключ '" ++ curKey ++ "' найден но продолжение
отсутствует(")
```

Исправленный код: Исправлена монада Maybe на Just

```
Just [] -> return (Left $ "Ключ '" ++ curKey ++ "' найден, но продолжение
отсутствует.")
```

Исходный код:

```
Nothing -> putStrLn "Ключ не найден !!!"
```

Исправленный код: добавлена проверка условий

```
Just -> putStrLn "Ключ не найден !!!"
Nothing -> putStrLn "Ключ не найден!»
```

9 Исправленный код

В коде к изменениям добавлены комментарии:

Lib.hs

```
{-# LANGUAGE LambdaCase #-}
module Lib (prodlojaNorm, nSlovarchik, saveSlovarchik, slovoKomputera, dialog) where
import Data.Char (isLetter, toLower)
import Data.List (nub, intercalate)
import Data.List.Split (splitWhen)
import Data.Map.Strict (Map)
import qualified Data.Map.Strict as Map
import System.Random (randomRIO)

prodlojaNorm :: String -> Maybe [[String]]
prodlojaNorm text
    | null text = Nothing
    | otherwise = Just $ filter (not . null) $ map (words . normPredloja) $ predlojaS
text
  where
    razdelit :: String
    razdelit = ".!?;:()"
    predlojaS :: String -> [String]
    predlojaS t = splitWhen (`elem` razdelit) t
    normPredloja :: String -> String
    normPredloja = map toLower . filter (\c -> isLetter c || c == ' ')

nSlovarchik :: Int -> [[String]] -> Map String [String]
nSlovarchik n predlojaS =
  Map.filter (not . null) $
    Map.fromListWith (\new old -> nub (old ++ new)) $
      [ (unwords key, [value])
        | predlojNSlovar <- predlojaS
        , (key, value) <- slovarVPredloj n predlojNSlovar
      ]
  where
    slovarVPredloj :: Int -> [String] -> [([String], String)]
    slovarVPredloj n words =
      [ (key, unwords value)
        | i <- [0 .. length words - 1]
        , k <- [1 .. min (n - 1) (length words - i)]
        , let key = take k (drop i words)
        , let rest = drop (i + k) words
        , not (null rest)
        , l <- [1 .. min (n - k) (length rest)]
        , let value = take l rest
        , not (null value)
      ]++
      [ (key, "")
        | i <- [max 0 (length words - (n - 1)) .. length words - 1], i >= 0
        , k <- [1 .. min (n - 1) (length words - i)]
        , let key = take k (drop i words)
        , length key == k
        , drop (i + k) words == []
      ]

saveSlovarchik :: FilePath -> Map String [String] -> IO ()
saveSlovarchik filePath dict =
    let vhod = Map.toAscList dict
        filterVhod = filter (not . null . snd) vhod
        formattedvhod = map formatNorm filterVhod
    in writeFile filePath (unlines formattedvhod)
  where
    formatNorm (key, values) =
```

```
      let values' = nub values
          nePusto = if all null values' then [] else filter (not . null) values'
          valuesStr = intercalate ", " (map show nePusto)
      in key ++ " : [" ++ valuesStr ++ "]"

slovoKomputera :: Map String [String] -> [String] -> IO (Either String [String])
slovoKomputera dict keyWords = generate (unwords keyWords) [] maxLength
  where
    maxLength = 15
    minLength = 2
    -- Исправлена орфографическая ошибка
    generate _ acc 0
        | length acc >= minLength = return (Right (reverse acc))
        | otherwise = return (Left "Ошибка!")
    generate curKey acc n =
        case Map.lookup curKey dict of
            Nothing
                | length acc >= minLength -> return (Right (reverse acc))
                | otherwise -> return (Left $ "Завершен!")
            -- Исправлена монада Maybe на Just и текст ошибки
            Just [] -> return (Left $ "Ключ '" ++ curKey ++ "' найден, но продолжение
отсутствует.")
            Just nextOptions ->
                randomRIO (0, length nextOptions - 1) >>= \idx ->
                let nextValue = nextOptions !! idx
                in if null nextValue
                   then return (Right (reverse acc))
                   else let continuation = words nextValue
                        in generate (unwords $ drop 1 (words curKey ++ continuation))
(continuation ++ acc) (n - 1)

keyPoUbiv :: [String] -> Map String [String] -> Maybe String
keyPoUbiv [] _ = Nothing
keyPoUbiv (x:xs) dict
    | Map.member x dict = Just x
    | otherwise = keyPoUbiv xs dict

dialog :: Map String [String] -> Map String [String] -> [String] -> Int -> IO ()
dialog dict1 dict2 initialPhrase steps =
    let logDialog prevPhrase d1 d2 remainingSteps sobesNumber
            | remainingSteps == 0 = putStrLn "Диалог завершён"
            | otherwise =
                let currentsobes = if sobesNumber `mod` 2 == 1 then 1 else 2
                    dict = if currentsobes == 1 then d1 else d2
                    otherDict = if currentsobes == 1 then d2 else d1
                    sobesLabel = "Чел " ++ show currentsobes ++ ": "
                in slovoKomputera dict prevPhrase >>= \case
                    Left err -> putStrLn (sobesLabel ++ err) >> putStrLn "Диалог
завершён"
                    Right [] -> putStrLn (sobesLabel ++ "Ошибочка генерации!(") >>
putStrLn "Диалог завершён"
                    Right generatedPhrase ->
                        let phraseWithInitial = unwords prevPhrase ++ " " ++ unwords
generatedPhrase
                            lastWords = reverse (words phraseWithInitial)
                        in putStrLn (sobesLabel ++ "(Ключ: " ++ unwords prevPhrase ++
") " ++ phraseWithInitial) >>
                           case keyPoUbiv lastWords otherDict of
                               -- добавлена проверка условий (исправление)
                               Just _ -> putStrLn "Ключ не найден !!!"
                               Nothing -> putStrLn "Ключ не найден!"
                               Just nextKey -> let nextPhrase = words nextKey
                                               in logDialog nextPhrase d1 d2
(remainingSteps - 1) (sobesNumber + 1)
    in logDialog initialPhrase dict1 dict2 steps 1
```

Заключение

В ходе работы был изучен метод ручного тестирования программного - инспекция кода. Было проведено инспекционное заседание, по результатам которого составлен протокол заседания. На основе протокола были внесены изменения в программный код.

В основном в ходе заседания были выявлены лексические ошибки в именах переменных, ошибки в форматировании данных, а также ошибки, связанные с областями видимости.

Метод ручного тестирования — инспекция кода — оказался полезным, поскольку позволил выявить ошибки, которые ранее были не замечены самим программистом.

Список литературы

1. Майерс, Г . Искусство тестирования программ. - Санкт-Петербург: Диалектика, 2012. - С.272.