ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «СИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ И ИНФОРМАТИКИ»

Кафедра вычислительных систем

КУРСОВАЯ РАБОТА

по дисциплине «Технологии разработки программного обеспечения» на тему «Quiz Runner». Система проведения тестов

Выполнил:

ст. гр. ИП-014

Обухов Артём Игоревич

Проверил:

доц., к.ф.-м.н.

Пудов Сергей Григорьевич

Содержание

[**Введение и постановка задачи** 2](#_Toc73447426)

[**Техническое задание** 3](#_Toc73447427)

[**О проекте** 3](#_Toc73447428)

[**Функционал проекта** 3](#_Toc73447429)

[**Формат входных данных** 4](#_Toc73447434)

[**Интерфейс приложения** 4](#_Toc73447435)

[**Взаимодействие с приложением** 5](#_Toc73447440)

[**Исходные файлы извне программы** 5](#_Toc73447441)

[**Описание выполненного проекта** 6](#_Toc73447442)

[**Личный вклад в проект** 11](#_Toc73447443)

[**Приложение. Текст программы** 12](#_Toc73447444)

# **Введение и постановка задачи**

Целью данной работы было освоить курс по предмету «технологии разработки программного обеспечения», который включает в себя использование системы контроля версий git, системы сборки приложения make, CI и юнит-тестирование, а также работу в команде

Задача — реализовать приложение для проведения психологического тестирования по методологии Big Five на языке C++ с системой контроля версий git, сделать сборку приложения с помощью makefile, подключить CI, а также покрыть код юнит-тестами.

# **Техническое задание**

## **О проекте**

QuizRunner представляет собой интерактивное приложение для проведения тестирования на ваш психологический портрет, разработанное при помощи низкоуровнего языка программирования С++ с использованием кроссплатформенной мультимедийной библиотеки SFML. Приложение будет поддерживаться на Linux.

## **Функционал проекта**

При запуске приложения перед пользователем будет появляться интерактивное меню:

### Начать глобальное тестирование

Тестирование, состоящее из 75-ти вопросов. На выбор предлагается 5 вариантов ответа и на каждый вопрос ответы одинаковы. При прохождении теста выводится полная картина вашего психологического портрета и личностных качеств в процентном соотношении.

### Тестирование по выбору

Тестирование на определенные личностные качества человека:

1. Нейротизм
2. Экстраверсия
3. Открытость опыту
4. Доброжелательность
5. Добросовестность

Прохождение тестирования по выбору содержит в себе 15 вопросов. Вопросы для тестирования по выбору берутся из тех же вопросов, которые предоставлены в глобальном тестировании, с той лишь разницей, что их меньше и все они будут относиться к архетипу, который был выбран. Формат проведения тестирования по выбору не отличается от глобального тестирования. Тестирование по выбору подходит для тех, кто:

* Не хочет тратить много времени на прохождение Глобального теста
* Хочет узнать свою предрасположенность к определенному личностному качеству

### Случайное тестирование

Случайным образом выбирается один из 5-ти вариантов, предложенных в тестировании по выбору.

### Выход

Выход из программы.

## **Формат входных данных**

Выбор ответов на вопросы среди предложенных, в последствии формирующих результат.

## **Интерфейс приложения**

Элементами интерфейса приложения будут выступать непосредственно пункты меню, ячейки для выбора варианта ответа, кнопки для перехода/возврата на следующую/ предыдущую страницу теста.

### Начать глобальное тестирование

При выборе данного пункта меню, пользователь незамедлительно начинает прохождение теста. Если случайно был отмечен нежелательный вариант ответа, то ответ можно очистить, выбрав другой вариант ответа. При успешном прохождения тестирования формируется результат, из полученных программой ответов, и выдается для ознакомления пользователю.

### Тестирование по выбору

При выборе данного пункта меню открывается еще одно меню с пунктами, на какое именно личностное качество пользователь желает пройти тестирование. Тестирование так же сопровождается всеми теми элементами интерфейса, перечисленными ранее.

### Случайное тестирование

При выборе данного пункта меню пользователя случайным образом бросает на один из пяти тестов по каждому личностному качеству.

### Выход

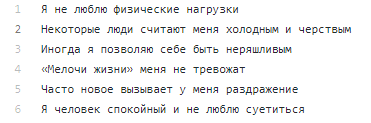
При выборе данного пункта меню пользователь незамедлительно закрывает приложение.

## **Взаимодействие с приложением**

Непосредственное взаимодействие с приложением осуществляется при помощи компьютерной мыши. Нужные пункты, меню, ячейки, кнопки выбираются/активируются при нажатии ЛКМ.

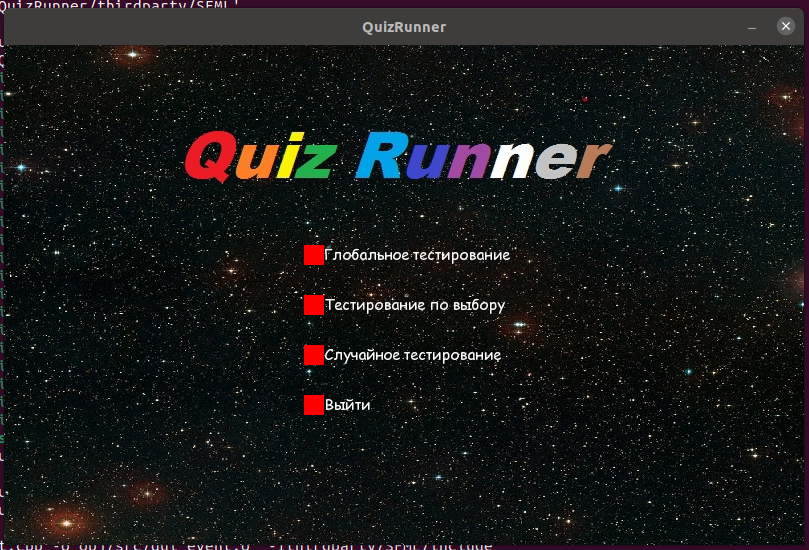
## **Исходные файлы извне программы**

Утверждения и вопросы будут записаны построчно в текстовом файле questions.txt в таком виде:



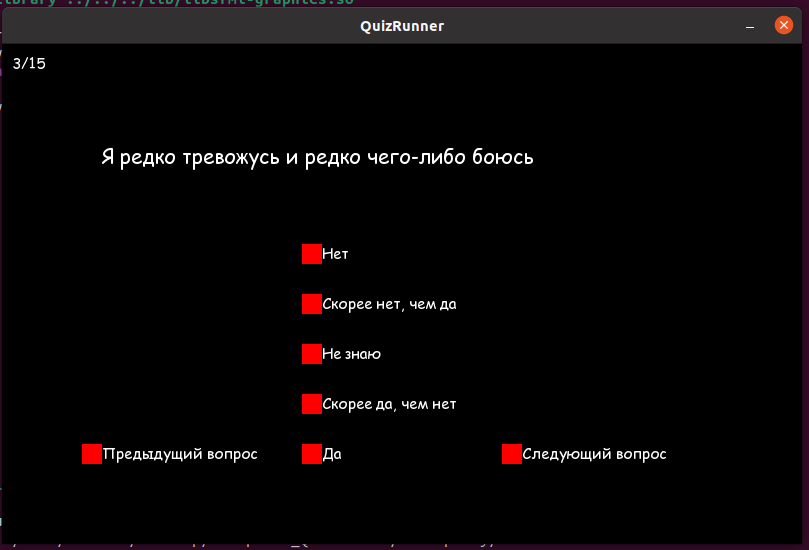
# **Описание выполненного проекта**

При запуске приложения открывается графическое меню из четырёх пунктов: глобальное тестирование, тестирование по выбору, случайное тестирование и выход из программы.

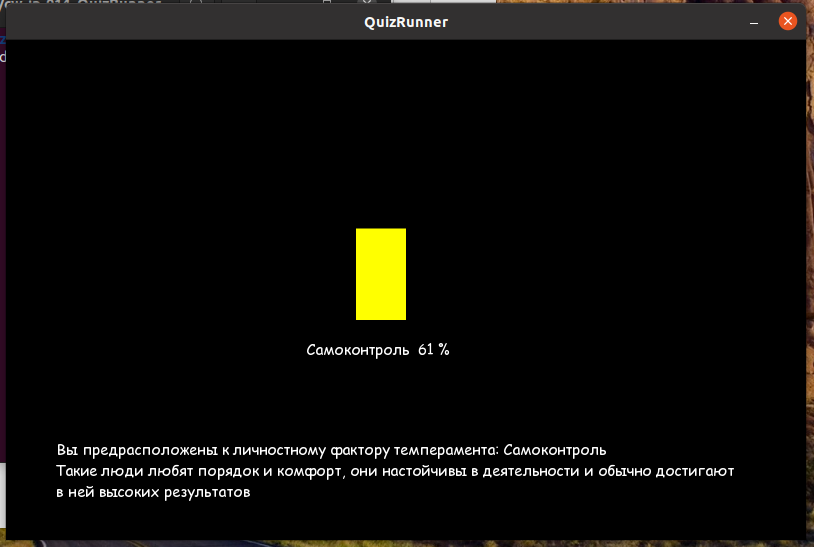


После выбора одного из трёх пунктов будут сформированы списки вопросов, по которым будет проводиться тестирование

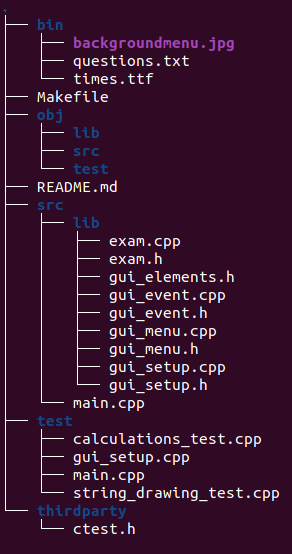
Окно с вопросом для ответа выглядит так:



После прохождения тестирования будут выведены результаты со шкалами и кратким диагностическим текстов на основе полученных в тесте результатов



**Структура проекта**



**Что было сделано:**

* Функция для считывания вопросов из текстового файла
* Основные графические элементы(виджеты) для реализации графического пользовательского интерфейса
* Функция, которая разделяет вопросы по категориям при тестировании по одной шкале
* Функция, которая обрабатывает нажатия мышью
* Функция, в которой происходит само тестирование
* Функция, которые вычисляют результат для отображения его в конце
* Функция, которая создает диагностический текст на основе результатов тестирования
* Разработано графическое меню с использованием ранее разработанных графических элементов
* Юнит-тестирование

**Функция для считывания вопросов из текстового файла**

* std::vector<JTextArea> generate\_questions\_list(sf::Font& font, std::ifstream& in) – производит считывание всех вопросов из текстового файла.

**Основные графические элементы(виджеты) для реализации графического пользовательского интерфейса**

* struct JScaleMettle – графическое представление шкалы при отображения результатов в конце проведения психологического тестирования
* struct JTextArea – отвечает за отображения текста
* struct Button – реализация кнопки для взаимодействия с графическим пользовательским интерфейсом
* struct QuestionCounter – реализация счётчика вопросов, который будет отображаться во время проведения тестирования
* std::vector<Button> generate\_template\_list(sf::Font& font, const std::vector<std::string>& dictionary) – создает меню на основе массива dictionary
* sf::String draw\_russian(std::string line) – нужна для отображения русского языка при использовании графической мультимедийной библиотеки SFML

**Функция, которая разделяет вопросы по категориям при тестировании по одной шкале**

* std::vector <JtextArea> generate\_question\_list\_on\_one\_scale(std::vector <JtextArea> question\_list, int scale\_number) – производит создание списка вопросов по одной шкале на основе уже считанного.

**Функция, которая обрабатывает нажатия мышью**

* void event\_key\_press(sf::Event& event, sf::RenderWindow& window, int& x\_mouse\_position, int& y\_mouse\_position) – производит обработку нажатий клавиши “Escape”, и выдает координаты мыши, если она была нажата
* void key\_flip(sf::Event& event, bool flag, long unsigned int& ques, bool mode, bool press) – функция обрабатывает нажатия мышью по элементам типа struct Button, также осуществляет возврат к предыдущему или переход к следующему вопросу при нажатии соответствующих кнопок.
* bool press\_select\_button(std::vector<Button>& selection\_list, int I, int x\_mouse\_position, int y\_mouse\_position, Button& next\_slide) – осуществляет взаимодействие с вариантами ответов, а также блокирует кнопку для перехода на следующий вопрос, если вариант ответа не был выбран

**Функция, в которой происходит само тестирование**

* void exam(int mode, int scale) – основная функция тестирования, где идёт использование всех предыдущих элеметов

**Функция, которые вычисляют результат для отображения его в конце**

* void add\_score\_scale(Button button, std::vector<int> scale\_list, int mode, int scale) – осуществляет начисление баллов за ответ на вопрос в тестировании в зависимости от указанной шкалы и режима тестирования
* int calculate\_percent(int score) – осуществляет подсчёт результата в процентах.
* std::string get\_string\_with\_percents(int score) – возвращает строку с процентами, посчитанными с использованием calculate\_percent(int score)
* std::vector<JScaleMettle> generate\_scale\_list(sf::Font& font, std::vector<int> score\_list) – реализует графическое представление результатов тестирования в виде графических шкал

**Функция, которая создает диагностический текст на основе результатов тестирования**

* JTextArea generate\_diagnostic\_text(sf::Font& font, std::vector<int> score\_list) – осуществляет создание диагностического текста на основе score\_list

**Разработано графическое меню с использованием ранее разработанных графических элементов**

* void menu() – осуществляет создание и отрисовки основного меню, где представлены основные четыре пункта(см. ТЗ)
* void temper\_menu() – осуществляет отрисовку подменю при выборе варианта “Тестирование по выбору”

**Юнит-тестирование**

* Нами было покрыты 8 функций из 15. С помощью библиотеки [CTEST](https://github.com/bvdberg/ctest)

# **Личный вклад в проект**

Непосредственно мой вклад в проект состоит в направлении других участников команды, подготовки структуры проекта, написание makefile’a, подключение CI для осуществления сборки и проведения юнит-тестирования проекта, многочисленные исправления, декомпозиция кода, а также реализация некоторых графических элементов и функций.

Мною были написаны:

struct QuestionCounter – осуществление счётчика вопросов. Обновление счётчика происходит через метод void update(int number\_of\_current\_question)

int calculate\_percent(int score) – осуществляет подсчёт процента по шкале на основе score(75 баллов – максимум по одной шкале)

std::string get\_string\_with\_percents(int score) – возвращает строку с использованием int calculate\_percent(int score)

std::vector <JTextArea> generate\_question\_list\_one\_one\_scale(std::vector<JTextArea> question\_list, int scale number) – осуществляет создания нового списка вопросов на основе question\_list и заданному scale\_number

sf::String draw\_russian(std::string) – отрисовка кириллицы. При использовании SFML возникла проблема с отрисовкой русского языка – эта функция решает данную проблему

std::vector<JScaleMettle> generate\_scale\_list(sf::Font& font, std::vector<int> score\_list) – реализует графическое отображение графического результата с помощью структуры JScaleMettle. На вход принимает score\_list, где находятся результаты тестирования.

# **Приложение. Текст программы**

*//File Makefile*

CFLAGS := -Wall -Wextra -Werror -std=c++17

CPPFLAGS := -MMD

CXX := g++

TARGET := bin/quizrunner

SOURCES := $(wildcard src/\*.cpp)

LIBSOURCES := $(wildcard src/lib/\*.cpp)

LIBOBJ := $(patsubst src/lib/%.cpp, obj/src/%.o, $(LIBSOURCES))

LIB := obj/lib/quizrunnerlib.a

OBJ := $(patsubst src/%.cpp, obj/src/%.o, $(SOURCES))

TEST := $(wildcard test/\*.cpp)

TESTOBJ := $(patsubst test/%.cpp, obj/test/%.o, $(TEST))

TESTTARGET := bin/testquizrunner

CTEST := thirdparty/ctest.h

SFML := thirdparty/SFML

SFMLLIB := $(SFML)/lib

SFMLINCLUDE := $(SFML)/include

LIBS=-lsfml-graphics -lsfml-window -lsfml-system

all: $(TARGET)

run: $(TARGET)

    cd bin; export LD\_LIBRARY\_PATH=../$(SFMLLIB) && ./quizrunner

$(TARGET):  $(SFMLLIB) $(LIB) $(OBJ)

    $(CXX) $(CPPFLAGS) $(CFLAGS) -o $(TARGET) $(OBJ) -L. $(LIB) -L$(SFMLLIB) $(LIBS)

$(SFMLLIB):

    git submodule update --init --recursive

    cmake $(SFML)/CMakeLists.txt

    make -C $(SFML)

$(LIB): $(LIBOBJ)

    ar rcs $@ $^

obj/src/%.o: src/lib/%.cpp

    $(CXX) $(CPPFLAGS) $(CFLAGS) -c $< -o $@  -I$(SFMLINCLUDE)

obj/src/%.o: src/%.cpp

    $(CXX) $(CPPFLAGS) $(CFLAGS) -c  $< -o $@  -I$(SFMLINCLUDE) -Isrc/lib

test: $(TESTTARGET)

    cd bin; export LD\_LIBRARY\_PATH=../$(SFMLLIB) && ./testquizrunner

$(TESTTARGET): $(TESTOBJ) $(CTEST) $(LIB)

    $(CXX) $(CPPFLAGS) $(CFLAGS)  $(TESTOBJ) -o $@ -L. $(LIB) -I src/lib -Ithirdparty -L$(SFMLLIB) $(LIBS)

obj/test/%.o: test/%.cpp $(CTEST) $(LIB)

    $(CXX) $(CPPFLAGS) $(CFLAGS) -c $< -o $@ -Isrc/lib -Ithirdparty -I$(SFMLINCLUDE)

clean:

    find . -name "\*.d" -exec rm {} \;

    find . -name "\*.o" -exec rm {} \;

    find . -name "\*.a" -exec rm {} \;

    find . -name "quizrunner" -exec rm {} \;

    find . -name "testquizrunner" -exec rm {} \;

format:

    cd src; find . -name "\*.cpp" -exec clang-format -i {} \;

    cd src; find . -name "\*.h" -exec clang-format -i {} \;

    cd test; find . -name "\*.cpp" -exec clang-format -i {} \;

.PHONY: clean run test all format

*//File gui\_elements.h*

struct QuestionCounter {

    int total\_amount\_of\_questions;

    sf::Text text;

    QuestionCounter(

            int total\_amount\_of\_questions,

            sf::Font& font,

            int font\_size,

            int x,

            int y)

    {

        this->total\_amount\_of\_questions = total\_amount\_of\_questions;

        text = sf::Text("", font, font\_size);

        text.setFillColor(sf::Color::White);

        text.setPosition(x, y);

    }

    void update(int number\_of\_current\_question)

    {

        text.setString(

                std::to\_string(number\_of\_current\_question) + "/"

                + std::to\_string(total\_amount\_of\_questions));

    }

};

*//File gui\_setup.cpp*

int calculate\_percent(int score)

{

    return (score / 0.01 / 75);

}

*std::string get\_ string\_with\_percents(int score)*

std::string get\_string\_with\_percents(int score)

{

    return std::to\_string(calculate\_percent(score));

}

*//File gui\_setup.cpp*

std::vector<JTextArea> generate\_question\_list\_on\_one\_scale(

        std::vector<JTextArea> question\_list, int scale\_number)

{

    std::vector<JTextArea> new\_question\_list;

    int current\_question\_number = scale\_number;

    while (current\_question\_number < 75) {

        new\_question\_list.push\_back(question\_list[current\_question\_number]);

        current\_question\_number += 5;

    }

    return new\_question\_list;

}

*//File gui\_setup.cpp*

*sf::String draw\_russian(std::string line)*

sf::String draw\_russian(std::string line)

{

    return sf::String::fromUtf8(line.begin(), line.end());

}

std::vector<JScaleMettle> generate\_scale\_list(sf::Font& font, std::vector<int> score\_list)

std::vector<JScaleMettle>

generate\_scale\_list(sf::Font& font, std::vector<int> score\_list)

{

    std::vector<JScaleMettle> scale\_list;

    int scale\_number = 0;

    if (score\_list[EXTRAVERSION]) {

        scale\_list.push\_back(JScaleMettle(

                10,

                300,

                sf::Text(draw\_russian("Экстраверсия "), font, 15),

                sf::Color::Green,

                calculate\_percent(score\_list[EXTRAVERSION])));

        std::string score\_percent

                = get\_string\_with\_percents(score\_list[EXTRAVERSION]);

        scale\_list[scale\_number].text.setString(

                scale\_list[scale\_number].text.getString() + " " + score\_percent

                + " " + "%");

        scale\_number++;

    }

    if (score\_list[AGREEABLENESS]) {

        scale\_list.push\_back(JScaleMettle(

                150,

                300,

                sf::Text(draw\_russian("Привязанность "), font, 15),

                sf::Color::Magenta,

                calculate\_percent(score\_list[AGREEABLENESS])));

        std::string score\_percent

                = get\_string\_with\_percents(score\_list[AGREEABLENESS]);

        scale\_list[scale\_number].text.setString(

                scale\_list[scale\_number].text.getString() + " " + score\_percent

                + " " + "%");

        scale\_number++;

    }

    if (score\_list[CONSCIENTIOUSNESS]) {

        scale\_list.push\_back(JScaleMettle(

                300,

                300,

                sf::Text(draw\_russian("Самоконтроль "), font, 15),

                sf::Color::Yellow,

                calculate\_percent(score\_list[CONSCIENTIOUSNESS])));

        std::string score\_percent

                = get\_string\_with\_percents(score\_list[CONSCIENTIOUSNESS]);

        scale\_list[scale\_number].text.setString(

                scale\_list[scale\_number].text.getString() + " " + score\_percent

                + " " + "%");

        scale\_number++;

    }

    if (score\_list[NEUROTICISM]) {

        scale\_list.push\_back(JScaleMettle(

                450,

                300,

                sf::Text(draw\_russian("Эмоциональная неуст. "), font, 15),

                sf::Color::Blue,

                calculate\_percent(score\_list[NEUROTICISM])));

        std::string score\_percent

                = get\_string\_with\_percents(score\_list[NEUROTICISM]);

        scale\_list[scale\_number].text.setString(

                scale\_list[scale\_number].text.getString() + " " + score\_percent

                + " " + "%");

        scale\_number++;

    }

    if (score\_list[OPENNESS]) {

        scale\_list.push\_back(JScaleMettle(

                630,

                300,

                sf::Text(draw\_russian("Экспрессивность "), font, 15),

                sf::Color::Red,

                calculate\_percent(score\_list[OPENNESS])));

        std::string score\_percent

                = get\_string\_with\_percents(score\_list[OPENNESS]);

        scale\_list[scale\_number].text.setString(

                scale\_list[scale\_number].text.getString() + " " + score\_percent

                + " " + "%");

        scale\_number++;

    }

    return scale\_list;

}