МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра информационных систем

КУРСОВАЯ РАБОТА

по дисциплине «Объектно-ориентированное программирование»

Тема: Программная система, имитирующая во времени процессы для вычислительного центра

Задание 11

Студенты гр. 8363 _____ Нерсисян А.С.

_____ Панфилович А.И.

Преподаватель _____ Егоров С.С.

Санкт-Петербург

2021

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ

Студенты Нерсисян А.С.

Панфилович А.И.

Группа 8363

Тема работы: программная система, имитирующая во времени процессы для вычислительного центра

Исходные данные:

В вычислительном центре работает 5 персональных компьютера (ПК). ПК отказывают с одинаковой вероятностью, причем неисправный ПК отказать не может. Неисправность проявляется в момент ее возникновения, но для определения ПК с неисправностью инженер должен периодически последовательно обходить работающие ПК и делать диагностику. При определении неисправности инженер прекращает обход и выполняет ремонт. Закончив его, инженер продолжает обход.

Содержание пояснительной записки:

«Задание на курсовую работу», «Содержание», «Введение», «Постановка задачи». «Описание предметной области», «Перечень библиотечных конструкторских классов, использованных в проекте (наименование, назначение)», «Обоснование выбора языка программирования и объектных библиотек конструкторских классов (параграф необходим, если используются другие, не рекомендованные для применения данным документом)», «Подсистема «Интерфейс»», «Графическое представление интерфейсных окон», «Основное окно», «Окно управления событиями ПрО», «Окно отображения состояния объектов ПрО», «Подсистема «Модель»», «Модель «сущность-связь» ПрО», «Перечень событий, изменяющих состояние ПрО», «Объектная модель», «Диаграмма классов», «Логическое описание полей

классов (тип, наименование, область видимости)», «Логическое описание методов классов (наименование, параметры вызова, область видимости)», «Заголовочные файлы (h-файлы) классов», «Диаграмма объектов ПрО», «Заключение», «Список использованных источников», «Приложение А. Листинг программного кода».

Предполагаемый объем пояснительной записки:

Не менее 40 страниц.

Дата выдачи задания: 01.03.2021

Дата сдачи курсовой работы: 27.05.2021

Дата защиты курсовой работы: 27.05.2021

Студенты _____ Нерсисян А.С.

_____ Панфилович А.И.

Преподаватель _____ Егоров С.С.

АННОТАЦИЯ

В данной работе имитируется жизненный цикл работы вычислительного центра. В вычислительном центре работает 5 персональных компьютера (ПК). ПК отказывают с одинаковой вероятностью, причем неисправный ПК отказать не может. Неисправность проявляется в момент ее возникновения, но для определения ПК с неисправностью инженер должен периодически последовательно обходить работающие ПК и делать диагностику. При определении неисправности инженер прекращает обход и выполняет ремонт. Закончив его, инженер продолжает обход. На всех этапах состояния ПК и действия инженера отображаются в окне состояний.

SUMMARY

This work simulates the life cycle of a computing center. The computing center has 5 personal computers (PCs). PCs fail with equal probability, and a defective PC cannot fail. A malfunction manifests itself at the time of its occurrence, but in order to identify a PC with a malfunction, the engineer must periodically sequentially bypass the running PCs and make diagnostics. When a fault is identified, the engineer stops bypassing and repairs. Having finished it, the engineer continues his round. At all stages, the state of the PC and the actions of the engineer are displayed in the state window.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В данной работе имитируется жизненный цикл работы вычислительного центра. В вычислительном центре работает 5 персональных компьютера (ПК). ПК отказывают с одинаковой вероятностью, причем неисправный ПК отказать не может. Неисправность проявляется в момент ее возникновения, но для определения ПК с неисправностью инженер должен периодически последовательно обходить работающие ПК и делать диагностику. При определении неисправности инженер прекращает обход и выполняет ремонт. Закончив его, инженер продолжает обход. На всех этапах состояния ПК и действия инженера отображаются в окне состояний.

В работе описывается предметная область задачи, перечисляются библиотечные конструкторские классы, использованные в проекте, также есть описание подсистемы интерфейс, модели сущность-связь, объектной модели (диаграммы классов).

1. ПОСТАНОВКА ЗАДАЧИ

В вычислительном центре работает 5 персональных компьютера (ПК). ПК отказывают с одинаковой вероятностью, причем неисправный ПК отказать не может. Неисправность проявляется в момент ее возникновения, но для определения ПК с неисправностью инженер должен периодически последовательно обходить работающие ПК и делать диагностику. При определении неисправности инженер прекращает обход и выполняет ремонт. Закончив его, инженер продолжает обход.

1.1 Описание предметной области

Предметной областью в данной работе является вычислительный центр. ПК находится в вычислительном центре, инженер работает в вычислительном центре и обслуживает ПК.

1.2 Перечень библиотечных конструкторских классов, использованных в проекте (наименование, назначение)

1.2.1 Класс Application

Класс Application хранит в себе указатели на интерфейс программы (interface), вычислительный центр (model) и инженера (enginer), а также связывает ключевые сигналы, генерируемые в классах interface, model и enginer с соответствующими слотами для обработки и корректного функционирования программы. Наследуется от библиотечного класса QApplication.

1.2.2 Класс Model

Класс Model описывает вычислительный центр (далее ВЦ), хранит в себе параметры (ParamData parameters), согласно которым идет имитация жизненного цикла ВЦ, и состояния (StateData state), в которых бывают объекты (ПК, инженер) в ВЦ. Наследуется от библиотечного класса QObject.

1.2.3 Класс Enginer

Класс Enginer представляет собой инженера. Реализует обслуживание (обход, диагностика, ремонт) ПК в ВЦ согласно заданным параметрам в соответствующем окне приложения. Хранит в себе состояния всех ПК в ВЦ (работает, не работает, диагностика, ремонт) и собственное состояние (свободен, занят обходом, занят диагностикой, занят ремонтом).

1.2.4 Классы Interface, StateWindow, ParamWindow, ControlWindow

Классы Interface, StateWindow, ParamWindow и ControlWindow описывают и реализовывают работу интерфейсных окон приложения.

2. ПОДСИСТЕМА «ИНТЕРФЕЙС»

Подсистема «интерфейс» состоит из четырех взаимосвязанных окон.

2.1 Графическое представление интерфейсных окон

На рисунках 1-4 представлены интерфейсные окна подсистем «Интерфейс», «Параметры», «Состояние» и «Управление».

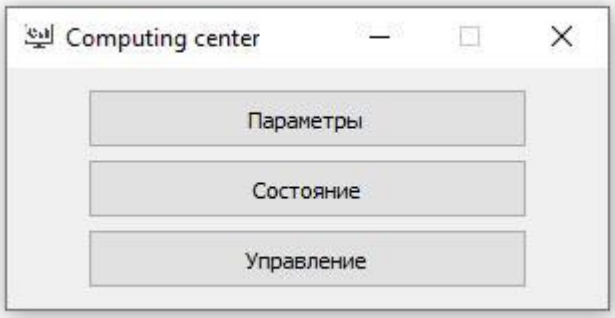

Рисунок 1 – Интерфейс программной системы

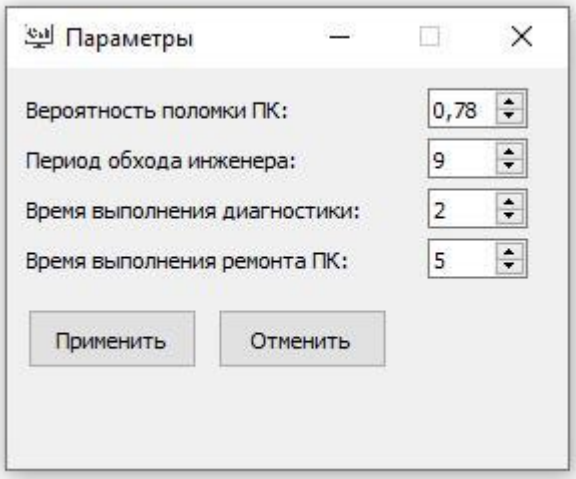

Рисунок 2 – Окно «Параметры»

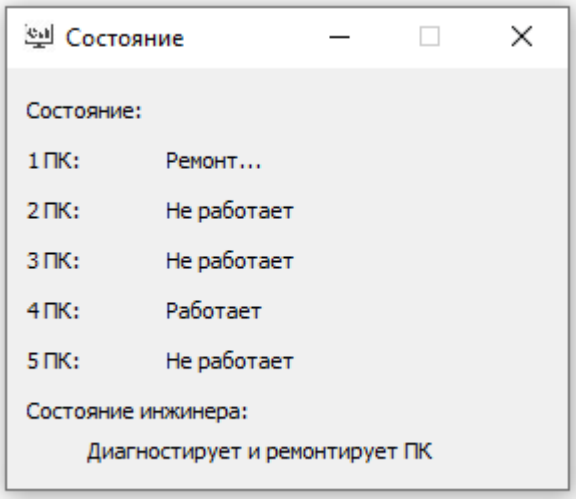

Рисунок 3 – Окно «Состояние»

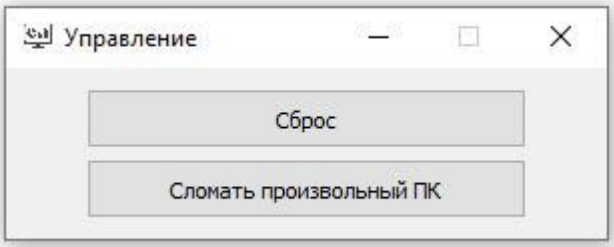

Рисунок 4 – Окно «Управление»

2.1.1 Основное окно

Основное окно программы содержит три кнопки, которые вызывают соответствующие дополнительные окна параметров, состояния и управления (см. рис. 1).

2.1.2 Окно управления событиями ПрО

Окно «Управление» содержит две кнопки (см. рис. 4):

- «Сброс» – сбрасывает состояние объектов до состояния по умолчанию;
- «Сломать произвольный ПК» -- инициирует поломку произвольного ПК, учитывая заданную вероятность поломки в окне параметров. ПК не сломается никогда, если значение вероятности поломки установлен в 0.00.

2.1.3 Окно отображения состояния объектов ПрО

Окно «Состояние» отображает текущее состояние объектов (см. рис.3)

Состояний всего 4:

- Работает
- Не работает
- Диагностика…
- Ремонт…

Состояния объектов меняются согласно описанной модели поведения.

2.1.4 Окно задания параметров объектов ПрО

Окно «Параметры» задает следующие параметры для генерации состояний объектов (см. рис.2).

Окно содержит 4 поле ввода и две кнопки:

Поля ввода:

- Вероятность поломки ПК – задает вероятность поломки произвольного ПК, входные значения ограничены в диапазоне чисел с плавающей точкой $[0.0; 1.0]$, где при 0 никогда ничего не сломается, а при 1 – с каждым тактом сломается одно ПК, в результате 5 тактов сломается вся система.
- Период обхода инженера – задает период обхода инженера в секундах, входные значения ограничены в диапазоне целых $[1; 100]$.
- Время выполнения диагностики – задает время выполнения диагностики в секундах, входные значения ограничены в диапазоне целых $[1; 100]$.
- Время выполнения ремонта ПК – задает время выполнения ремонта ПК в секундах, входные значения ограничены в диапазоне целых $[1; 500]$.

Кнопки:

- «Применить» – применяет введенные параметры;
- «Отменить» – сбрасывает последние примененные параметры.

3. ПОДСИСТЕМА «МОДЕЛЬ»

Подсистема «Модель» меняет состояния объектов согласно заданным параметрам поведения.

3.1 Модель «сущность-связь» ПрО

Модель «сущность-связь» ПрО продемонстрирована на рисунке 5. Данная система предполагает наличие трех сущностей: ПК, инженер, очередь обхода и обслуживания. Каждая из сущностей имеет связь друг с другом:

1) ПК находятся в вычислительном центре (M:1)

2) Инженер работает в вычислительном центре (1:1);

2) Инженер обслуживает ПК в вычислительном центре (1:1);

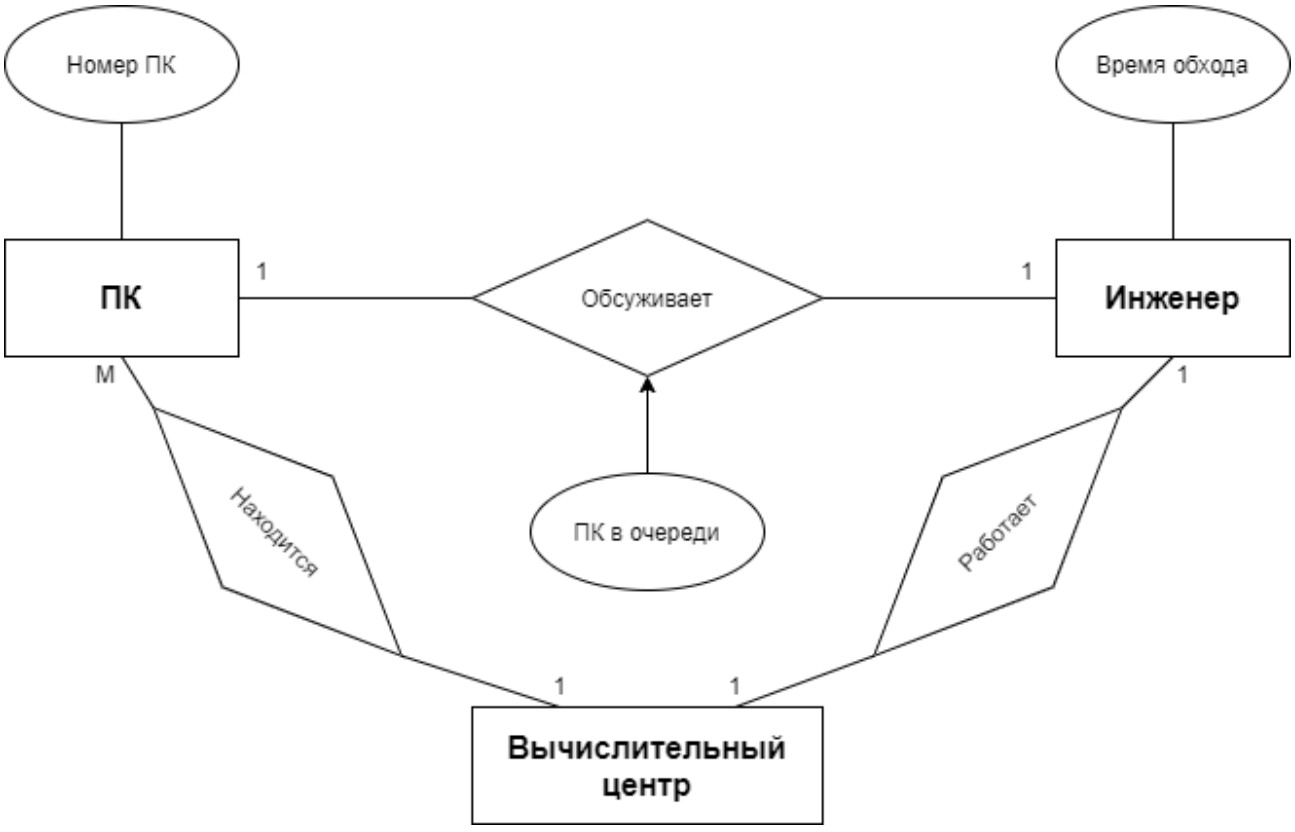

Рисунок 5 – Модель «сущность-связь» ПрО

3.2 Перечень событий, изменяющих состояние ПрО

Состояние объектов меняют следующие события:

- сигналы, генерируемые по нажатию интерфейсных кнопок;
- сигналы, генерируемые по таймеру.

1. Связь между классами Interface - Model
   1.1. Сигнал: SIGNAL(sendInterfaceEvent(Events))
   Слот: SLOT(recieveModelEvent(Events))
   1.2. Сигнал: SIGNAL(sendModelEvent(Events))
   Слот: SLOT(recieveInterfaceEvent(Events))
2. Связь между классами Interface - Enginer
   2.1. Сигнал: SIGNAL(sendInterfaceEvent(Events))
   Слот: SLOT(receiveEnginerEvent(Events))
3. Связь между классами Enginer - Model
   3.1. Сигнал: SIGNAL(sendEnginerEvent(Events))
   Слот: SLOT(recieveModelEvent(Events))
   3.2. Сигнал: SIGNAL(sendModelEvent(Events))
   Слот: SLOT(receiveEnginerEvent(Events))
4. Связь между кнопками и соответствующими действиями для класса Interface
   4.1. Сигнал: pbtn, SIGNAL(pressed())
   Слот: interface, SLOT(openParamWindow())
   4.2. Сигнал: sbtn, SIGNAL(pressed())
   Слот: interface, SLOT(openStateWindow())
   4.3. Сигнал: cbtn,SIGNAL(pressed())
   Слот: interface, SLOT(openControlWindow())
5. При нажатии на всевозможные кнопки, находящиеся в интерфейсных окнах в классах ParamWindow, StateWindow и ControlWindow сгенерированные сигналы pressed(), перенаправляются в класс Interface и выполняется сигнал

sendInterfaceEvent(Events msg), который отправляет сигнал в класс Model в слот recieveModelEvent(Events msg).

6. Сигналы timeout(), которые генерируются по таймеру (интервал зависит от заданных параметров в окне «Параметры», класс ParamWindow), отправляются в следующие слоты: SayEnginerToCheck(); SayEnginerToDiag(); SayEnginerToRepair(); SayEnginerToFix(). Они в нужный момент сообщают инженеру (объект класса Enginer), что порам менять состояния объектов ПК в ВЦ.

4. ОБЪЕКТНАЯ МОДЕЛЬ

4.1 Диаграмма классов

Диаграмма классов показывает набор классов и их связи, как показано на рисунке 6.

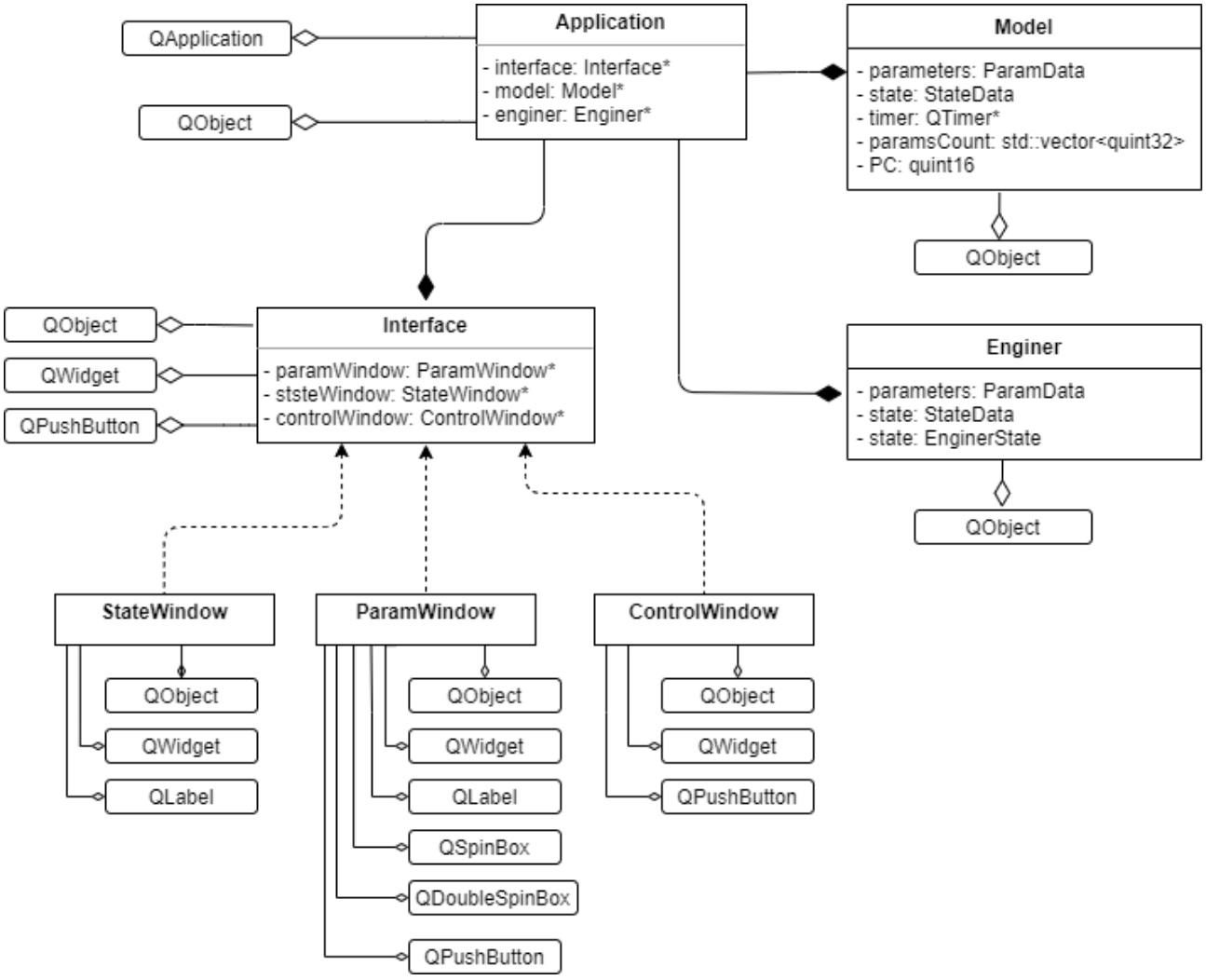

Рисунок 6 – Диаграмма классов ПрО

Класс «Application» − представляет собой класс, отвечающий за работу программы, является точкой входа в программу.

Класс «Model» − представляет собой класс, отвечающий за реализацию объектной модели предметной области, который находится в отношении композиции с классом «Application».

Класс «Enginer» − представляет собой класс, отвечающий за реализацию работы инженера, находится в отношении композиции с классом «Application».

Класс «Interface» − представляет собой класс, отвечающий за вывод основного окна, который находится в отношении композиции с классом «Application».

Классы «StateWindow», «ParamWindow» и «ControlWindow» − представляют собой класс составные части класса «Interface», находятся в отношении зависимости с классом «Interface».

4.1.1 Логическое описание полей классов (тип, наименование, область видимости)

Класс Application

Тип	Наименование	Область видимости
Interface*	interface	private
Model*	model	private
Enginer*	enginer	private

Класс Model

Тип	Наименование	Область видимости
ParamData	parameters	private
StateData	state	private
QTimer*	timer	private
std::vector<quint32>	paramsCount	private
quint16	PC	private

Класс Interface

Тип	Наименование	Область видимости
ParamWindow*	paramWindow	private
StateWindow	stateWindow	private
ControlWindow*	controlWindow	private
QPushButton*	pbtn	private
QPushButton*	sbtn	private
QPushButton*	cbtn	private

Класс ParamWindow

Тип	Наименование	Область видимости
QLabel*	l1	private
QLabel*	l2	private
QLabel*	l3	private
QLabel*	l4	private
QDoubleSpinBox*	crashProbability	private
QSpinBox*	checkPeriod	private
QSpinBox*	diagnosticsTime	private
QSpinBox*	repairTime	private
QPushButton*	apply_btn	private
QPushButton*	cancel_btn	private

Класс StateWindow

Тип	Наименование	Область видимости
QLabel*	stateLabel	private
QLabel*	labelPC1	private
QLabel*	labelPC2	private
QLabel*	labelPC3	private
QLabel*	labelPC4	private
QLabel*	labelPC5	private
QLabel*	statePC1	private
QLabel*	statePC2	private
QLabel*	statePC3	private
QLabel*	statePC4	private
QLabel*	statePC5	private
QLabel*	labelEnginer	private
QLabel*	enginerState	private

Класс ControlWindow

Тип	Наименование	Область видимости
QPushButton*	b1	private
QPushButton *	b2	private

4.1.2 Логическое описание методов классов (наименование, параметры вызова, область видимости)

Класс Model

Наименование	Параметры вызова	Область видимости
init	-	private
tact	-	private
paramRequest	-	private
stateRequest	-	private

Класс Interface

Наименование	Параметры вызова	Область видимости
closeEvent	QCloseEvent*	protected

Класс ParamWindow

Наименование	Параметры вызова	Область видимости
setCurrentParams	l1	public
closeEvent	QCloseEvent*	protected

Класс StateWindow

Наименование	Параметры вызова	Область видимости
setCurrentState	const StateData&	public
closeEvent	QCloseEvent*	protected

Класс ControlWindow

Наименование	Параметры вызова	Область видимости
closeEvent	QCloseEvent*	protected

4.1.3 Заголовочные файлы (h-файлы) классов

- application.h
- model.h
- enginer.h
- interface.h
- paramwindow.h
- statewindow.h
- controlwindow.h

ЗАКЛЮЧЕНИЕ

В данной работе была проектирована и реализована программа, имитирующая жизненный цикл работы вычислительного центра. На всех этапах состояния ПК и действия инженера отображаются в окне состояний.

В работе описана предметная область задачи, перечислены библиотечные конструкторские классы, использованные в проекте, также есть описание подсистемы интерфейс, модели сущность-связь, объектной модели (диаграммы классов).

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Объектно-ориентированное программирование в C++. Классика Computer Science. 4-е издание / Р.Лафоре СПб., Питер, 2016. 928 с.

2. Техническая документация по Qt. // Qt Documentation URL: https://doc.qt.io/ (дата обращения: 26.05.2021).

3. Документация по C++. // CPP Reference. URL: https://en.cppreference.com/w/ (дата обращения: 17.05.2021).

4. Документация по C++. // C++ Reference URL: https://www.cplusplus.com/reference/ (дата обращения: 01.04.2021).

5. Документация по Microsoft C++ // Техническая документация Майкрософт для разработчиков. URL: https://docs.microsoft.com/ru-ru/cpp/?view=msvc-160 (дата обращения: 22.05.2021).

ПРИЛОЖЕНИЕ А. ЛИСТИНГ ПРОГРАММНОГО КОДА

Листинг 1. file CW.pro

```
QT       += core gui
greaterThan(QT_MAJOR_VERSION, 4): QT += widgets

TARGET = OOP_CourceWork
CONFIG += c++11
TEMPLATE = app

DEFINES += QT_DEPRECATED_WARNINGS

SOURCES += \
    application.cpp \
    controlwindow.cpp \
    enginer.cpp \
    main.cpp \
    interface.cpp \
    model.cpp \
    paramwindow.cpp \
    statewindow.cpp

HEADERS += \
    application.h \
    controlwindow.h \
    enginer.h \
    eventtypes.h \
    interface.h \
    model.h \
    paramdata.h \
    paramwindow.h \
    statedata.h \
    statewindow.h
```

Листинг 2. file main.cpp

```
#include "interface.h"
#include "application.h"

int main(int argc, char *argv[])
{
    Application a(argc, argv);
    return a.exec();
}
```

Листинг 3. file application.h

```
#ifndef APPLICATION_H
#define APPLICATION_H

#include <QApplication>
#include <QObject>
#include "interface.h"
#include "model.h"
#include "enginer.h"

class Application : public QApplication
{
    Q_OBJECT

    Interface  *interface;
    Model      *model;
    Enginer    *enginer;

public:
    Application(int,char**);
    ~Application();
};

#endif // APPLICATION_H
```

Листинг 4. file application.cpp

```
#include "application.h"
#include <QThread>

Application::Application(int argc, char** argv) :
    QApplication(argc,argv)
{
    interface = new Interface();
    interface->show();
```

```
    enginer = new Enginer();
    model = new Model(enginer);

    connect(interface,SIGNAL(sendInterfaceEvent(Events)),
            model,SLOT(recieveModelEvent(Events)));
    connect(model,SIGNAL(sendModelEvent(Events)),
            interface,SLOT(recieveInterfaceEvent(Events)));

    connect(interface,SIGNAL(sendInterfaceEvent(Events)),
            enginer,SLOT(receiveEnginerEvent(Events)));

    connect(enginer, SIGNAL(sendEnginerEvent(Events)),
            model, SLOT(recieveModelEvent(Events)));
    connect(model, SIGNAL(sendModelEvent(Events)),
            enginer, SLOT(receiveEnginerEvent(Events)));
}


Application::~Application()
{
    delete model;
    delete interface;
    delete enginer;
}
```

Листинг 5. file model.h

```
#ifndef MODEL_H
#define MODEL_H

#include <QObject>
#include <QTimer>
#include "enginer.h"
#include "eventtypes.h"

class Model : public QObject
{
private:
    Q_OBJECT

    static ParamData defaultParameters;
    static StateData defaultState;

    ParamData parameters;
    StateData state;

    Enginer* enginer;

    QTimer *timer;
```

```
    std::vector<quint32> paramsCount;
    quint16 PC;

    void init();
    void tact();
    void paramRequest();
    void stateRequest();

public:
    Model(Enginer*);
    ~Model();

signals:
    void sendModelEvent(Events);

public slots:
    void recieveModelEvent(Events);
    void SayEnginerToCheck();
    void SayEnginerToDiag();
    void SayEnginerToRepair();
    void SayEnginerToFix();

};

#endif // MODEL_H
```

Листинг 6. file model.cpp

```
#include "model.h"
#include <algorithm>
#include <random>
#include <chrono>
#include <windows.h>
#include <QThread>



ParamData Model::defaultParameters = {0.5,5,1,7};
StateData Model::defaultState      = {QString("Работает"),
                                      QString("Работает"),
                                      QString("Работает"),
                                      QString("Работает"),
                                      QString("Работает"),
                                      QString("Проверяет ПК с заддaным пе
риудом")};

Model::Model(Enginer* enginer_) : QObject(), timer(new QTimer()), paramsC
ount(4)
{
    parameters = defaultParameters;
```

```
    state = defaultState;
    enginer = enginer_;
    //enginer->setParamsAndState(state, parameters);
    timer->setInterval(1000); //1 sec interval
    connect(timer, SIGNAL(timeout()),this, SLOT(SayEnginerToCheck()));
    connect(timer, SIGNAL(timeout()),this, SLOT(SayEnginerToDiag()));
    connect(timer, SIGNAL(timeout()),this, SLOT(SayEnginerToRepair()));
    connect(timer, SIGNAL(timeout()),this, SLOT(SayEnginerToFix()));
    //connect(timer, SIGNAL(timeout()),this, SLOT(SetEnginerState()));
    timer->start();

}

Model::~Model()
{
    delete timer;
}

void Model::recieveModelEvent(Events msg)
{
    switch (msg.type)
    {
        case PARAMREQUEST:
            paramRequest();
            break;
        case PARAMMESSAGE:
            parameters = msg.p;
            break;
        case STATEREQUEST:
            stateRequest();
            break;
        case STATEMESSAGE:
            state = msg.s;
            stateRequest();
            break;
        case RESET:
            init();
            paramRequest();
            stateRequest();
            break;
        case TACT:
            tact();
            stateRequest();
            break;
        case ENGINERSTATEMESSAGE: {
            state = msg.s;
            if (msg.PC >= 1 && msg.PC <= 5)
                PC = msg.PC;
            else PC = 0;
            Events msg2(STATEMESSAGE);
            msg2.s = state;
```

```
            //отправка на интерфейс
            emit sendModelEvent(msg2);
            break;
        }

        default: break;
    }
}



void Model::init()
{
    parameters = defaultParameters;
    state = defaultState;
    enginer->setParamsAndState(state, parameters);
}


void Model::tact()
{
    std::vector<int> a(100);
    int prob = int(parameters.crashProbability * 100);
    fill_n(a.begin(), prob, 1);             // 1 - сломаем
    fill_n(a.begin() + prob, 100-prob, 0); // 0 - не сломаем

    //перемешиваем
    unsigned seed = std::chrono::system_clock::now().time_since_epoch().c
ount();
    shuffle(a.begin(), a.end(), std::default_random_engine(seed));

    srand(time(NULL));
    //тыкаем в рандомную позицию
    bool isCrash = bool(a[int(abs(rand())%100)]);

    //сломаем или не сломаем
    if (isCrash) {
        do {
            //если все таки сломаем, то выбираем какой именно
            int id = int(abs(rand())%5);
            //если все не работают
            if (state.statePC1 == "Не работает" &&
                state.statePC2 == "Не работает" &&
                state.statePC3 == "Не работает" &&
                state.statePC4 == "Не работает" &&
                state.statePC5 == "Не работает")
            {
                //выходим из цикла ибо нечего сломать.
                break;
            }
            switch (id) {
```

```
                case 0:
                    //если уже сломанно, пытаемся еще раз выбрать рандомн
ый ПК
                    if(state.statePC1 == "Не работает")
                        continue;
                    state.statePC1 = "Не работает";
                    break;
                case 1:
                    if(state.statePC2 == "Не работает")
                        continue;
                    state.statePC2 = "Не работает";
                    break;
                case 2:
                    if(state.statePC3 == "Не работает")
                        continue;
                    state.statePC3 = "Не работает";
                    break;
                case 3:
                    if(state.statePC4 == "Не работает")
                        continue;
                    state.statePC4 = "Не работает";
                    break;
                case 4:
                    if(state.statePC5 == "Не работает")
                        continue;
                    state.statePC5 = "Не работает";
                    break;
                default:
                    break;
            }
            //если все таки вышли из свич блока
            //значит дело сделано и бесконечный цикл
            //можно прервать
            break;
        }while (true);
    }
    //enginer->setParamsAndState(state, parameters);
}

void Model::paramRequest()
{
    Events msg(PARAMMESSAGE);
    msg.p = parameters;
    emit sendModelEvent(msg);
}

void Model::stateRequest()
{
    Events msg(STATEMESSAGE);
    msg.s = state;
    emit sendModelEvent(msg);
```

```
}

void Model::SayEnginerToCheck()
{
    paramsCount[0]++;
    if (paramsCount[0] >= parameters.checkPeriod &&
        enginer->getEnginerState() == NOTBUSY ) {
        //если инжинер НЕ занят обходом, диагностикой или ремонтом
        Events msg(CHECK);
        msg.s = state;
        msg.p = parameters;
        paramsCount[0] = 0;
        emit sendModelEvent(msg);
    }
}

void Model::SayEnginerToDiag()
{
    paramsCount[1]++;
    if (!(PC >= 1 && PC <= 5)) return;
    quint32 time = parameters.checkPeriod;
    if (paramsCount[1] >= time &&
            enginer->getEnginerState() == CHECKWORK){
        Events msg(DIAG);
        msg.s = state;
        msg.PC = PC;
        paramsCount[1] = 0;
        emit sendModelEvent(msg);
    }
}

void Model::SayEnginerToRepair()
{
    paramsCount[2]++;
    if (!(PC >= 1 && PC <= 5)) return;
    quint32 time = parameters.checkPeriod +
                   parameters.diagnosticsTime;
    if (paramsCount[2] >= time &&
            enginer->getEnginerState() == DIAGWORK){
        Events msg(REPAIR);
        msg.s = state;
        msg.PC = PC;
        paramsCount[2] = 0;
        emit sendModelEvent(msg);
    }
}

void Model::SayEnginerToFix()
{
    paramsCount[3]++;
    if (!(PC >= 1 && PC <= 5)) return;
```

```
    quint32 time = parameters.checkPeriod +
                   parameters.diagnosticsTime +
                   parameters.repairTime;
    if (paramsCount[3] >= time &&
            enginer->getEnginerState() == REPAIRWORK){
        Events msg(FIX);
        msg.s = state;
        msg.PC = PC;
        emit sendModelEvent(msg);
        paramsCount[3] = 0;
    }
}
```

Листинг 7. file interface.h

```
#ifndef INTERFACE_H
#define INTERFACE_H

#include <QWidget>
#include <QCloseEvent>
#include <QPushButton>

#include "eventtypes.h"

class ParamWindow;
class StateWindow;
class ControlWindow;

class Interface : public QWidget
{
private:
    Q_OBJECT

    ParamWindow   *paramWindow;
    StateWindow   *stateWindow;
    ControlWindow *controlWindow;

    QPushButton *pbtn;
    QPushButton *sbtn;
    QPushButton *cbtn;

public:
    Interface(QWidget *parent = 0);
    ~Interface();

protected:
    void closeEvent(QCloseEvent*);

signals:
    void sendInterfaceEvent(Events);
```

```
public slots:
    void paramWindowClosed();
    void stateWindowClosed();
    void controlWindowClosed();
    void recieveInterfaceEvent(Events);

private slots:
    void openParamWindow();
    void openStateWindow();
    void openControlWindow();
};

#endif // INTERFACE_H
```

Листинг 8. file interface.cpp

```
#include "interface.h"
#include "paramwindow.h"
#include "statewindow.h"
#include "controlwindow.h"

Interface::Interface(QWidget *parent)
    : QWidget(parent)
{
    setFixedSize(300,120);
    setWindowTitle("Computing center");

    paramWindow   = nullptr;
    stateWindow   = nullptr;
    controlWindow = nullptr;

    pbtn = new QPushButton("Параметры",this);
    pbtn->setGeometry(40,10,220,30);
    connect(pbtn,SIGNAL(pressed()),this,SLOT(openParamWindow()));

    sbtn = new QPushButton("Состояние",this);
    sbtn->setGeometry(40,45,220,30);
    connect(sbtn,SIGNAL(pressed()),this,SLOT(openStateWindow()));

    cbtn = new QPushButton("Управление",this);
    cbtn->setGeometry(40,80,220,30);
    connect(cbtn,SIGNAL(pressed()),this,SLOT(openControlWindow()));
}

Interface::~Interface()
{
    delete cbtn;
    delete pbtn;
    delete sbtn;
```

```
}

void Interface::recieveInterfaceEvent(Events msg)
{
    switch (msg.type)
    {
        case PARAMMESSAGE:
            if (paramWindow != nullptr)
                paramWindow->setCurrentParams(msg.p);
            break;
        case STATEMESSAGE:
            if (stateWindow != nullptr)
                stateWindow->setCurrentState(msg.s);
            break;
        default: break;
    }
}

void Interface::closeEvent(QCloseEvent* event)
{
    if (paramWindow != nullptr)
    {
        disconnect(paramWindow,SIGNAL(closing()),
                this,SLOT(paramWindowClosed()));
        paramWindow->close();
    }
    if (stateWindow != nullptr)
    {
        disconnect(stateWindow,SIGNAL(closing()),
                this,SLOT(stateWindowClosed()));
        stateWindow->close();
    }
    if (controlWindow != nullptr)
    {
        disconnect(controlWindow,SIGNAL(closing()),
                this,SLOT(controlWindowClosed()));
        controlWindow->close();
    }
    event->accept();
}

void Interface::paramWindowClosed()
{
    disconnect(paramWindow,SIGNAL(sendParamMessage(Events)),
               this,SIGNAL(sendInterfaceEvent(Events)));
    disconnect(paramWindow,SIGNAL(closing()),
               this,SLOT(paramWindowClosed()));
    paramWindow = nullptr;
}

void Interface::stateWindowClosed()
```

```
{
    disconnect(stateWindow,SIGNAL(closing()),
               this,SLOT(stateWindowClosed()));
    stateWindow = nullptr;
}

void Interface::controlWindowClosed()
{
    disconnect(controlWindow,SIGNAL(closing()),
               this,SLOT(controlWindowClosed()));
    disconnect(controlWindow,SIGNAL(sendControlMessage(Events)),
               this,SIGNAL(sendInterfaceEvent(Events)));
    controlWindow = nullptr;
}

void Interface::openParamWindow()
{
    if (paramWindow == nullptr)
    {
        paramWindow = new ParamWindow();
        connect(paramWindow,SIGNAL(closing()),
                this,SLOT(paramWindowClosed()));
        connect(paramWindow,SIGNAL(sendParamMessage(Events)),
                this,SIGNAL(sendInterfaceEvent(Events)));
        paramWindow->show();
        Events msg(PARAMREQUEST);
        emit sendInterfaceEvent(msg);
    }
}

void Interface::openStateWindow()
{
    if (stateWindow == nullptr)
    {
        stateWindow = new StateWindow();
        connect(stateWindow,SIGNAL(closing()),
                this,SLOT(stateWindowClosed()));
        stateWindow->show();
        Events msg(STATEREQUEST);
        emit sendInterfaceEvent(msg);
    }
}

void Interface::openControlWindow()
{
    if (controlWindow == nullptr)
    {
        controlWindow = new ControlWindow();
        connect(controlWindow,SIGNAL(closing()),
                this,SLOT(controlWindowClosed()));
        connect(controlWindow,SIGNAL(sendControlMessage(Events)),
```

```
                this,SIGNAL(sendInterfaceEvent(Events)));
        controlWindow->show();
    }
}
```

Листинг 9. file controlwindow.h

```
#ifndef CONTROLWINDOW_H
#define CONTROLWINDOW_H

#include <QWidget>
#include <QCloseEvent>
#include <QPushButton>
#include "eventtypes.h"

class ControlWindow : public QWidget
{
Q_OBJECT

    QPushButton *b1;
    QPushButton *b2;

public:
    ControlWindow(QWidget *parent = 0);
    ~ControlWindow();

protected:
    void closeEvent(QCloseEvent*);

private slots:
    void controlEvent();

signals:
    void closing();
    void sendControlMessage(Events);
};

#endif // CONTROLWINDOW_H
```

Листинг 10. file controlwindow.cpp

```
#include "controlwindow.h"

ControlWindow::ControlWindow(QWidget *parent) :
    QWidget(parent)
{
    setAttribute(Qt::WA_DeleteOnClose,true);
    setFixedSize(300,85);
    setWindowTitle("Управление");
```

```
    b1 = new QPushButton("Сброс",this);
    b1->setGeometry(40,10,220,30);
    connect(b1,SIGNAL(pressed()),this,SLOT(controlEvent()));

    //b2 = new QPushButton("Одиночное событие",this);
    b2 = new QPushButton("Сломать произвольный ПК",this);
    b2->setGeometry(40,45,220,30);
    connect(b2,SIGNAL(pressed()),this,SLOT(controlEvent()));
}

ControlWindow::~ControlWindow()
{
    delete b2;
    delete b1;
}

void ControlWindow::controlEvent()
{
    QPushButton *btn = (QPushButton*)sender();
    if (btn == b1)
    {
        Events msg(RESET);
        emit sendControlMessage(msg);
    }
    if (btn == b2)
    {
        Events msg(TACT);
        emit sendControlMessage(msg);
    }
}

void ControlWindow::closeEvent(QCloseEvent* event)
{
    emit closing();
    event->accept();
}
```

Листинг 11. file paramwindow.h

```
#ifndef PARAMWINDOW_H
#define PARAMWINDOW_H

#include <QWidget>
#include <QCloseEvent>
#include <QLabel>
#include <QSpinBox>
#include <QPushButton>

#include "eventtypes.h"
```

```
class ParamWindow : public QWidget
{
private:
    Q_OBJECT

    QLabel *l1;
    QLabel *l2;
    QLabel *l3;
    QLabel *l4;
    QDoubleSpinBox  *crashProbability;
    QSpinBox        *checkPeriod;
    QSpinBox        *diagnosticsTime;
    QSpinBox        *repairTime;
    QPushButton     *apply_btn;
    QPushButton     *cancel_btn;

public:
    ParamWindow(QWidget *parent = 0);
    ~ParamWindow();

    void setCurrentParams(const ParamData&);

protected:
    void closeEvent(QCloseEvent*);

protected slots:
    void setModelParam();
    void restoryModelParam();

signals:
    void closing();
    void sendParamMessage(Events);
};

#endif // PARAMWINDOW_H
```

Листинг 12. file paramwindow.cpp

```
#include "paramwindow.h"

ParamWindow::ParamWindow(QWidget *parent) :
    QWidget(parent)
{
    setAttribute(Qt::WA_DeleteOnClose,true);
    setWindowTitle("Параметры");
    setFixedSize(280,200);

    l1 = new QLabel("Вероятность поломки ПК:",this);
    l1->setGeometry(10,10,200,20);
```

```
    crashProbability = new QDoubleSpinBox(this);
    crashProbability->setRange(0.0,1.0);
    crashProbability->setGeometry(210,10,50,20);

    l2 = new QLabel("Период обхода инженера:",this);
    l2->setGeometry(10,35,200,20);
    checkPeriod = new QSpinBox(this);
    checkPeriod->setRange(1,100);
    checkPeriod->setGeometry(210,35,50,20);

    l3 = new QLabel("Время выполнения диагностики:",this);
    l3->setGeometry(10,60,200,20);
    diagnosticsTime = new QSpinBox(this);
    diagnosticsTime->setRange(1,100);
    diagnosticsTime->setGeometry(210,60,50,20);

    l4 = new QLabel("Время выполнения ремонта ПК:",this);
    l4->setGeometry(10,85,200,20);
    repairTime = new QSpinBox(this);
    repairTime->setRange(1,500);
    repairTime->setGeometry(210,85,50,20);

    apply_btn = new QPushButton("Применить",this);
    apply_btn->setGeometry(10,120,85,30);
    connect(apply_btn,SIGNAL(pressed()),this,SLOT(setModelParam()));

    cancel_btn = new QPushButton("Отменить",this);
    cancel_btn->setGeometry(105,120,85,30);
    connect(cancel_btn,SIGNAL(pressed()),this,SLOT(restoryModelParam()));
}

ParamWindow::~ParamWindow()
{
    delete l1;
    delete l2;
    delete l3;
    delete l4;
    delete crashProbability;
    delete checkPeriod;
    delete diagnosticsTime;
    delete repairTime;
    delete apply_btn;
    delete cancel_btn;
}

void ParamWindow::closeEvent(QCloseEvent* event)
{
    emit closing();
    event->accept();
}
```

```
void ParamWindow::setCurrentParams(const ParamData& pd)
{
    crashProbability->setValue(pd.crashProbability);
    checkPeriod->setValue(pd.checkPeriod);
    diagnosticsTime->setValue(pd.diagnosticsTime);
    repairTime->setValue(pd.repairTime);
}

void ParamWindow::setModelParam()
{
    ParamData pd;
    Events msg(PARAMMESSAGE);
    pd.crashProbability = crashProbability->value();
    pd.checkPeriod = checkPeriod->value();
    pd.diagnosticsTime = diagnosticsTime->value();
    pd.repairTime = repairTime->value();
    msg.p = pd;
    emit sendParamMessage(msg);
}

void ParamWindow::restoryModelParam()
{
    Events msg(PARAMREQUEST);
    emit sendParamMessage(msg);
}
```

Листинг 13. file statewindow.h

```
#ifndef STATEWINDOW_H
#define STATEWINDOW_H

#include <QWidget>
#include <QCloseEvent>
#include <QLabel>

#include "statedata.h"

class StateWindow : public QWidget
{
private:
    Q_OBJECT

    QLabel *stateLabel;
    QLabel *labelPC1;
    QLabel *labelPC2;
    QLabel *labelPC3;
    QLabel *labelPC4;
    QLabel *labelPC5;
    QLabel *statePC1;
    QLabel *statePC2;
```

```
    QLabel *statePC3;
    QLabel *statePC4;
    QLabel *statePC5;
    QLabel *labelEnginer;
    QLabel *enginerState;

public:
    StateWindow(QWidget *parent = 0);
    ~StateWindow();

    void setCurrentState(const StateData&);

protected:
    void closeEvent(QCloseEvent*);

signals:
    void closing();
};

#endif // STATEWINDOW_H
```

Листинг 14. file statewindow.cpp

```
#include "statewindow.h"
#include "interface.h"

StateWindow::StateWindow(QWidget *parent) :
    QWidget(parent)
{
    setAttribute(Qt::WA_DeleteOnClose,true);
    setFixedSize(280,210);
    setWindowTitle("Состояние");

    stateLabel = new QLabel("Состояние:",this);
    stateLabel->setGeometry(10,10,180,20);

    labelPC1 = new QLabel("1 ПК:",this);
    labelPC1->setGeometry(10,35,50,20);
    statePC1 = new QLabel(this);
    statePC1->setGeometry(80,35,130,20);

    labelPC2 = new QLabel("2 ПК:",this);
    labelPC2->setGeometry(10,60,50,20);
    statePC2 = new QLabel(this);
    statePC2->setGeometry(80,60,130,20);

    labelPC3 = new QLabel("3 ПК:",this);
    labelPC3->setGeometry(10,85,50,20);
    statePC3 = new QLabel(this);
    statePC3->setGeometry(80,85,130,20);
```

```
    labelPC4 = new QLabel("4 ПК:",this);
    labelPC4->setGeometry(10,110,50,20);
    statePC4 = new QLabel(this);
    statePC4->setGeometry(80,110,130,20);

    labelPC5 = new QLabel("5 ПК:",this);
    labelPC5->setGeometry(10,135,50,20);
    statePC5 = new QLabel(this);
    statePC5->setGeometry(80,135,130,20);

    labelEnginer = new QLabel("Состояние инжинера:", this);
    labelEnginer->setGeometry(10,160,160,20);
    enginerState = new QLabel(this);
    enginerState->setGeometry(40,180,230,20);
}

StateWindow::~StateWindow()
{
    delete stateLabel;
    delete labelPC1;
    delete labelPC2;
    delete labelPC3;
    delete labelPC4;
    delete labelPC5;
    delete statePC1;
    delete statePC2;
    delete statePC3;
    delete statePC4;
    delete statePC5;
    delete labelEnginer;
    delete enginerState;
}

void StateWindow::setCurrentState(const StateData& s)
{
    statePC1->setText(s.statePC1);
    statePC2->setText(s.statePC2);
    statePC3->setText(s.statePC3);
    statePC4->setText(s.statePC4);
    statePC5->setText(s.statePC5);
    enginerState->setText(s.enginerState);
    //repaint();
}

void StateWindow::closeEvent(QCloseEvent* event)
{
    emit closing();
    event->accept();
}
```

Листинг 15. file paramdata.h

```
#ifndef PARAMDATA_H
#define PARAMDATA_H

#include <QObject>

struct ParamData
{
    float crashProbability;     //0-1
    quint32 checkPeriod;        //seconds
    quint32 diagnosticsTime;    //seconds
    quint32 repairTime;         //seconds
};

#endif // PARAMDATA_H
```

Листинг 16. file statemdata.h

```
#ifndef STATEDATA_H
#define STATEDATA_H

#include <QString>

struct StateData
{
    QString statePC1;
    QString statePC2;
    QString statePC3;
    QString statePC4;
    QString statePC5;
    QString enginerState;
};

#endif // STATEDATA_H
```

Листинг 17. file eventtypes.h

```
#ifndef EVENTTYPES_H
#define EVENTTYPES_H

#include "paramdata.h"
#include "statedata.h"

enum EEvents
{
    CHECK = 1,
    DIAG,
    REPAIR,
    FIX,
```

```
    ENGINERSTATEMESSAGE,
    ENGINERDIAGMSG,
    PARAMREQUEST,
    PARAMMESSAGE,
    STATEREQUEST,
    STATEMESSAGE,
    RESET,
    TACT,
};

struct Events
{
    EEvents type;
    ParamData  p;
    StateData  s;
    quint16 PC;
    Events(EEvents t) { type = t; }
};

enum EnginerState {
    CHECKWORK = 1,
    DIAGWORK,
    REPAIRWORK,
    NOTBUSY,
};

#endif // EVENTTYPES_H
```