Санкт-Петербургский политехнический университет Петра Великого

Институт машиностроения, материалов и транспорта

Высшая школа автоматизации и робототехники

Курсовая работа

Дисциплина: Объектно-ориентированное программирование

Тема: Фреймворк Qt

Студент гр. 3331506/00401 Чанчиков Д.В.

Преподаватель Ананьевский М.С.

Санкт-Петербург

2023

Оглавление

1. Введение

Цель работы – изучение фреймворка Qt и его применение в разработке приложения с графическим интерфейсом.

Задачи:

1. Изучить основные принципы и концепции фреймворка Qt;
2. Изучить основные компоненты и модули, входящие в состав Qt;
3. Изучить инструменты и средства разработки, предоставляемые Qt;
4. Изучить способы интеграции Qt в различные языки программирования;
5. Разработать пример приложения, используя Qt, включающий основные функциональные возможности фреймворка, такие как создание интерфейса пользователя, обработку событий и другие;
6. Протестировать и отладить разработанное приложение, удостоверившись в его корректной работе и соответствии требованиям;
7. Сделать выводы о применимости и эффективности использования Qt в разработке программного обеспечения.

Помимо указанных задач, важно также продемонстрировать понимание основных принципов объектно-ориентированного программирования, а также умение применять и анализировать документацию и примеры кода, предоставляемые Qt.

2. Основная часть

2.1. Теоретическая часть

Фреймворк Qt (также известный как Qt Framework или просто Qt) представляет собой кроссплатформенный инструментарий разработки программного обеспечения, разработанный компанией The Qt Company. Он предоставляет разработчикам спектр функциональных возможностей и инструментов для создания приложений с графическим интерфейсом пользователя, а также других типов программного обеспечения.

Основные принципы и концепции фреймворка Qt включают:

1. Кроссплатформенность: Qt разработан с учетом кроссплатформенной совместимости, что означает, что приложения, созданные с использованием Qt, могут работать на разных операционных системах без изменений в исходном коде;
2. Объектно-ориентированное программирование: Qt использует принципы ООП, что облегчает структурирование и организацию кода. Он предоставляет классы и механизмы наследования, полиморфизма, инкапсуляции и другие;
3. Сигналы и слоты: это механизм взаимодействия между объектами в Qt. Сигналы представляют собой события, создаваемые объектом, а слоты являются функциями, которые вызываются в ответ на эти сигналы. Это обеспечивает реакции на события и обновление состояние интерфейса;
4. Модульность: Qt состоит из различных модулей, которые предоставляют функциональность в определенных областях. Это позволяет разработчикам выбирать и использовать только необходимые модули, что уменьшает размер приложения и повышает его эффективность;
5. Визуальное программирование: Qt предоставляет инструменты, позволяющие разработчикам создавать пользовательский интерфейс визуально. Это упрощает процесс проектирования пользовательского интерфейса и ускоряет разработку приложений.

Далее более подробно рассмотрим модульность фреймворка Qt. Ниже приведены некоторые из основных компонентов и возможностей Qt:

1. Qt Core: основной модуль, предоставляющий базовые классы и функции, включая контейнеры данных, обработку событий, управление памятью, работу с файлами и строками, механизмы сигналов и слотов;
2. Qt GUI: модуль для создания графического интерфейса пользователя (GUI), содержащий элементы управления, окна, рисование, обработку событий, работу с графикой и векторной графикой;
3. Qt Widgets: набор предопределенных элементов управления, таких как кнопки, текстовые поля, таблицы, меню, панели инструментов и др., для создания настольных приложений;
4. Qt Network: модуль для работы с сетевыми протоколами, клиент-серверным взаимодействием, передачей данных по сети, работой с HTTP, FTP и др.;
5. Qt SQL: модуль, предоставляющий классы и функции для работы с базами данных, поддерживая различные СУБД, такие как SQLite, MySQL и др.;
6. Qt XML: модуль для работы с XML-данными, включая чтение, запись и обработку XML-файлов.

Помимо основных компонентов, существуют и другие важные модули и возможности в фреймворке Qt, однако в рамках данной работы будут использоваться преимущественно первые 3 модуля.

Также обратимся к визуальности фреймворка: один из инструментов, предоставляемых Qt, называется Qt Designer. Он позволяет разработчикам создавать пользовательский интерфейс визуально без необходимости написания кода. Разработчики могут создавать формы, на которых могут размещать и настраивать элементы управления, такие как кнопки, поля ввода, таблицы, меню и другие, изменять их размеры и расположение, применять стили и настраивать их свойства, такие как цвет, текст, шрифт и т.д. Другой важной особенностью является возможность соединения сигналов и слотов между элементами управления. Разработчик может установить соединения между событиями

(сигналами), которые генерируются элементами управления, и функциями (слотами), которые должны быть вызваны в ответ на эти события.

Основным языком программирования для разработки с использованием Qt является C++, Разработчики могут использовать Qt API в своих C++ проектах для создания кроссплатформенных приложений с графическим интерфейсом, обработки событий, доступа к базам данных и т. д. Однако данный фреймворк предоставляет механизмы и инструменты для работы с разными языками программирования, что позволяет разработчикам выбирать наиболее подходящий язык для своих проектов. Так, интеграция с Python осуществляется с помощью библиотеки PyQt или PySide, интеграция с Java – с помощью библиотеки Qt Jambi. Qt также поддерживает интеграцию с другими языками программирования, такими как Ruby, JavaScript, Rust и другими. Существуют соответствующие библиотеки и привязки, которые позволяют разработчикам использовать Qt в проектах, написанных на этих языках.

Некоторые теоретические аспекты, не представленные выше, будут дополнительно рассмотрены в практической части с конкретными примерами использования.

2.2. Практическая часть

Пример приложения с графическим интерфейсом на основе фреймворка Qt был выполнен по техническому заданию, полученному от преподавателя технологий конструкционных материалов.

Техническое задание – необходимо создать приложение, которое рассчитывает некоторые параметры, необходимые для проектирования устройства под названием «гидросъемник». Набор входных и выходных параметров расчета заранее не известен и определяется пользователем индивидуально при каждом использовании программы. Основное окно разделено на 4 зоны: 3 зоны содержат соответственно названия и значения технологических параметров, геометрических параметров и параметров материалов, в оставшейся зоне размещен чертеж устройства.

Дизайн приложения создан с помощью инструмента Qt Designer. На рисунке 1 представлено основное окно «Ввод данных», на рисунке 2 – окно результатов «Полученные значения» окно сообщения об ошибке «Ошибка» и окно предупреждения «Предупреждение».

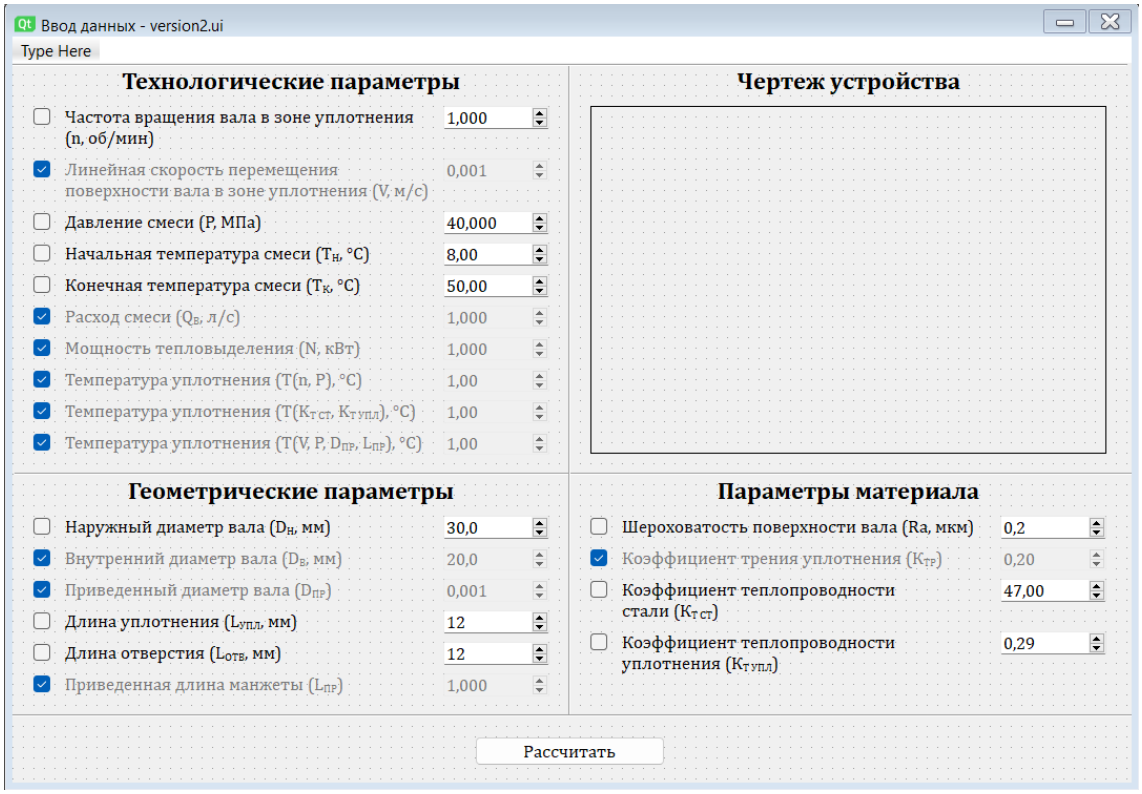

Рисунок 1 – Дизайн основного окна «Ввод данных»

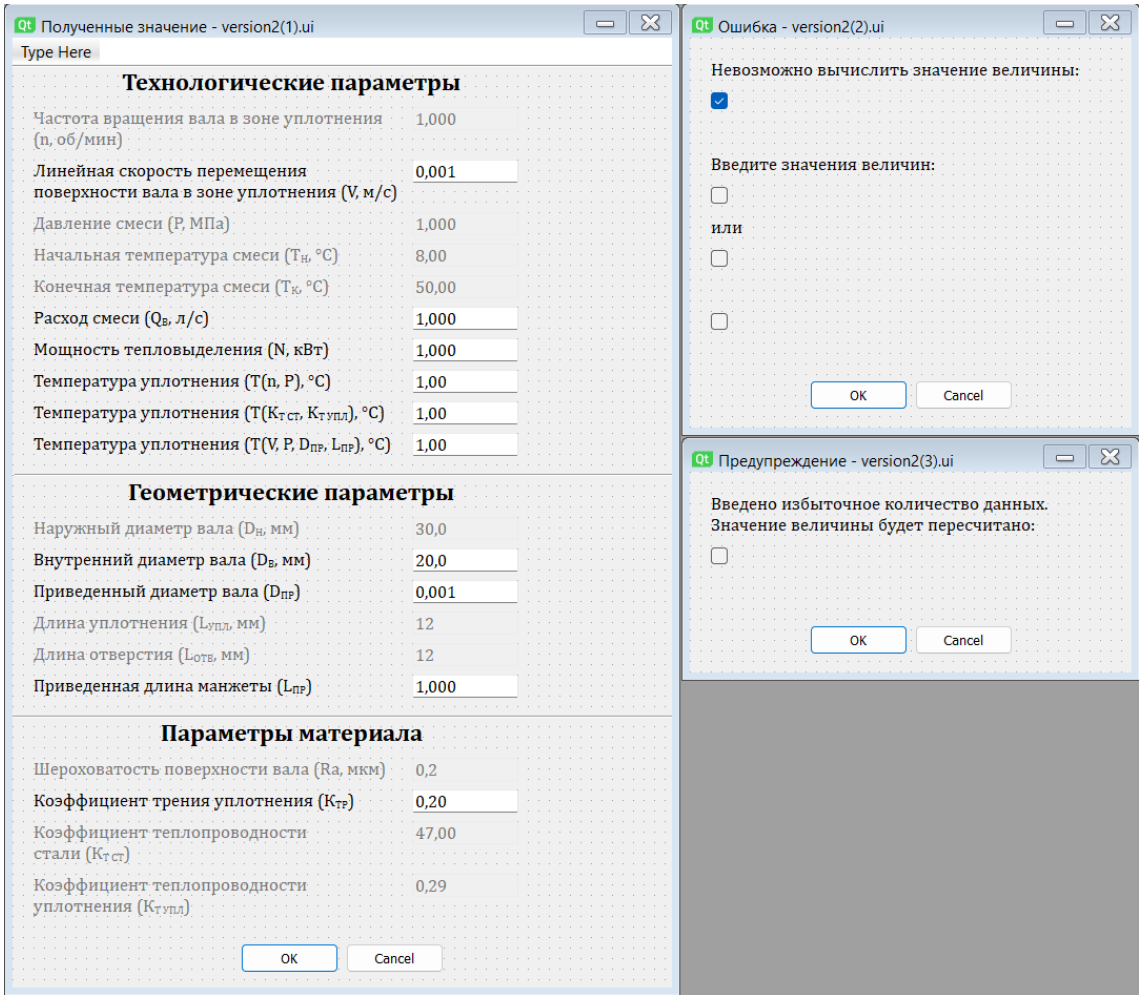

Рисунок 2 – Дизайн окна результатов «Полученные значения», окна сообщения об ошибке «Ошибка» и окна предупреждения «Предупреждение»

На данном этапе были отмечены флажками параметры, которые будут вычисляться программой по умолчанию. Также для каждого поля ввода чисел «doubleSpinBox» были установлены число знаков после запятой, минимальное и максимальное значения, шаг изменения значения и значение по умолчанию. Пример приведен на рисунке 3.

decimals	2
minimum	8.000000
maximum	20.000000
singleStep	0.500000
value	8.000000

Рисунок 3 – Пример установки ограничений поля ввода чисел «doubleSpinBox»

Изображение устройства, а также сигналы и слоты добавлялись не в инструменте Qt Designer, а позже, в самом программном коде.

В качестве языка программирования был выбран Python. Его преимущество перед C++ в данном проекте основано на 2-х факторах:

1. В программе отсутствуют сложные и объемные вычисления, рекурсии, соответственно отсутствует необходимость в работе с памятью и высокой производительности приложения;
2. Python обладает более простым и понятным синтаксисом, а также известен своей читаемостью и лаконичностью, что способствует более быстрой и простой разработке кода.

Для интеграции фреймворка Qt в Python использовалась библиотека PyQt. Она предоставляет полный доступ к Qt API, включая Qt Widgets, Qt Core, Qt Network, Qt SQL и другие модули Qt. Библиотека PyQt также обеспечивает интеграцию с Qt Designer, позволяя визуально проектировать пользовательский интерфейс и сохранять его в файле формата .ui. Затем этот файл формы можно загрузить в приложение на Python с использованием PyQt и связать элементы управления с функциональностью приложения.

Описанный выше алгоритм был выполнен для всех 3-х окон приложения. Таким образом, были получены 3 отдельных файла в формате .py, полностью описывающие окно и все его элементы (пример представлен на рисунке 4). Далее эти файлы были импортированы в основной файл проекта (полный код представлен в приложении).

```
import subprocess
subprocess.run(["pyuic5", "-x", "version2.ui", "-o", "version2.py"])
```

Рисунок 4 – Пример генерации кода в формате .py из файла в формате .ui

Структура программы выглядит следующим образом (рисунок 5): импортируются необходимые библиотеки и модули, а также файлы, описывающие дизайн окон, далее определяются классы окон, внутри которых определяются их методы и события, а затем создается блок "__main__", который будет непосредственно выполняться при запуске программы.

```
import math
from PyQt5 import QtCore
from PyQt5 import QtWidgets
from PyQt5.QtGui import QPixmap, QIcon
from PyQt5.QtWidgets import QApplication, QMainWindow, QDialog, QAction
from version2 import Ui_MainWindow
from version21 import Ui_NewWindow
from version22 import Ui_Error
from version23 import Ui_Warning

class MainWindow(QMainWindow, Ui_MainWindow):...

class NewWindow(QMainWindow, Ui_NewWindow):...

class Error(QDialog, Ui_Error):...

class Warning(QDialog, Ui_Warning):...

if __name__ == "__main__":...
```

Рисунок 5 – Структура программы

Далее рассмотрим структуру каждого класса. На рисунке 6 приведена структура класса MainWindow.

```
class MainWindow(QMainWindow, Ui_MainWindow):

    # Конструктор класса
    def __init__(self):...

    # Функция для снятия флажка с переданного элемента
    def change_with_element(self, element):...

    # Функция для смены установки флажков (RadioButton)
    def change_flag(self, state, element):...

    # Функция для связи события и действия по названию
    def connect_checkbox(self, name):...

    # Функция для смены активности элементов (1 - не акт, 0 - акт)
    def set_elements_disenabled(self, state, *elements):...

    # Функция для получения значений из активных элементов
    def get_value_if_element_enabled(self):...

    # Функция для расчета и вывода в новое окно
    def calculate(self):...
```

Рисунок 6 – Структура класса MainWindow

В конструкторе данного класса происходит инициализация и настройка элементов интерфейса, которые не были реализованы в Qt Designer, связывание сигналов и слотов, а также создание экземпляров дополнительных окон:

1. Вызывается конструктор родительского класса и метод, который устанавливает пользовательский интерфейс;
2. Устанавливается иконка окна, изображение чертежа и фокусы элементов;
3. Создается словарь var_dict, который содержит названия переменных в качестве ключей и их значения, инициализированные нулем;
4. Настраиваются связи сигналов и слотов для различных элементов;
5. Создаются экземпляры классов NewWindow, Error, Warning и передаются ссылки на соответствующие текущие экземпляры.

Далее определяются методы (функции) для обработки различных событий:

1. change_with_element(self, element): Этот метод снимает флажок с переданного элемента, если ранее на нем был установлен флажок;
2. change_flag(self, state, element): Этот метод используется для переключения состояния флажка на противоположное;
3. connect_checkbox(self, name): Этот метод по имени флажка в качестве аргумента устанавливает связь между событием изменения состояния флажка и методом set_elements_disenabled;
4. set_elements_disenabled(self, state, *elements): Этот метод изменяет активность переданных элементов, основываясь на переданном состоянии;
5. get_value_if_element_enabled(self): Этот метод возвращает словарь значений активных элементов интерфейса;
6. calculate(self): Этот метод вызывает get_value_if_element_enabled для получения актуальных значений, затем на их основе выполняет ряд расчетов, передает полученный словарь со значениями в новое окно и отображает его.

На рисунках 7 – 9 приведены соответственно примеры связи сигналов и слотов для различных элементов, примеры определения некоторых методов, структура метода calculate(self).

```python
# Частота и лин. скорость не могут быть выбраны одновременно (но могут быть не выбраны одновременно)
self.checkBox_chastota.clicked.connect(lambda: self.change_with_element(self.checkBox_lin_skoroct))
self.checkBox_lin_skoroct.clicked.connect(lambda: self.change_with_element(self.checkBox_chastota))

# При изменении шероховатости устанавливается флажок в коэфф. трения и наоборот
self.checkBox_sheroh.clicked.connect(lambda state: self.change_flag(state, self.checkBox_trenie_uplotn))
self.checkBox_trenie_uplotn.clicked.connect(lambda state: self.change_flag(state, self.checkBox_sheroh))

# Следующие величины не могут быть отмечены флажками
self.checkBox_nach_temp.clicked.connect(lambda: self.checkBox_nach_temp.setCheckState(QtCore.Qt.Unchecked))
self.checkBox_konech_temp.clicked.connect(lambda: self.checkBox_konech_temp.setCheckState(QtCore.Qt.Unchecked))
"""..."""

# Следующие величины отмечены флажками, которые нельзя снять
self.checkBox_rashod_vody.clicked.connect(lambda: self.checkBox_rashod_vody.setCheckState(QtCore.Qt.Checked))
self.checkBox_moshnoct_tepl.clicked.connect(lambda: self.checkBox_moshnoct_tepl.setCheckState(QtCore.Qt.Checked)
"""..."""
```

Рисунок 7 – Примеры связи сигналов и слотов для различных элементов

```python
# Функция для связи события и действия по названию
def connect_checkbox(self, name):
    checkbox = getattr(self, f"checkBox_{name}")
    label = getattr(self, f"label_{name}")
    spinbox = getattr(self, f"doubleSpinBox_{name}")
    checkbox.stateChanged.connect(lambda state: self.set_elements_disenabled(state, label, spinbox))

# Функция для смены активности элементов (1 - не акт, 0 - акт)
def set_elements_disenabled(self, state, *elements):
    for element in elements:
        element.setEnabled(not state)
```

Рисунок 8 – Примеры определения некоторых методов

```python
# Функция для расчета и вывода в новое окно
def calculate(self):

    # Обновление словаря
    updated_dict = self.get_value_if_element_enabled()
    self.var_dict.update(updated_dict)

    # Рассчет параметров переменной активности
    if self.var_dict["davlenie"] != 0:...
    else:...
    if self.var_dict["sheroh"] != 0:...
    else:...

    # Рассчет остальных параметров
    self.var_dict["diam_priv"] = self.var_dict["diam_vnutr"] / self.var_dict["diam_nar"]
    self.var_dict["dlina_priv"] = self.var_dict["dlina_otv"] / self.var_dict["dlina_uplotn"]
    """..."""

    # Выгрузка значений в новое окно и его вывод
    self.new_window.set_value_and_change_enable(self.var_dict)
    self.new_window.show()
```

Рисунок 9 – Структура метода calculate(self)

Структура остальных классов NewWindow, Error, Warning аналогична структуре класса MainWindow, хотя их функционал гораздо меньше. В блоке "__main__" создается экземпляр приложения QApplication и основное окно MainWindow, а затем главный цикл app.exec_() начинает обрабатывать события, позволяя взаимодействовать с пользователем в приложении (рисунок 10).

```
class NewWindow(QMainWindow, Ui_NewWindow):
    # Конструктор класса
    def __init__(self, main_window):...

    # Функция для вывода значений в новое окно
    def set_value_and_change_enable(self, values_dict):...

class Error(QDialog, Ui_Error):
    # Конструктор класса
    def __init__(self, parent=None):...

class Warning(QDialog, Ui_Warning):
    # Конструктор класса
    def __init__(self, parent=None):...

if __name__ == "__main__":
    app = QApplication([])
    window = MainWindow()
    window.show()
    app.exec_()
```

Рисунок 10 – Структура классов NewWindow, Error, Warning; блок "__main__"

Завершающим шагом в создании программы является реализация исполняемого файла .exe с помощью следующей команды в терминале:

pyinstaller --noconsole **hydro.py**

На следующих рисунках 11 – 14 приведены примеры работы программы в различных режимах. Первые 2 окна соответствуют стандартному (штатному) режиму работы. Последние 2 окна служат для того, чтобы сообщить пользователю о возникновении непредвиденных ситуаций, когда невозможно вычислить некоторое значение или когда некоторое значение может быть вычислено не единственным образом.

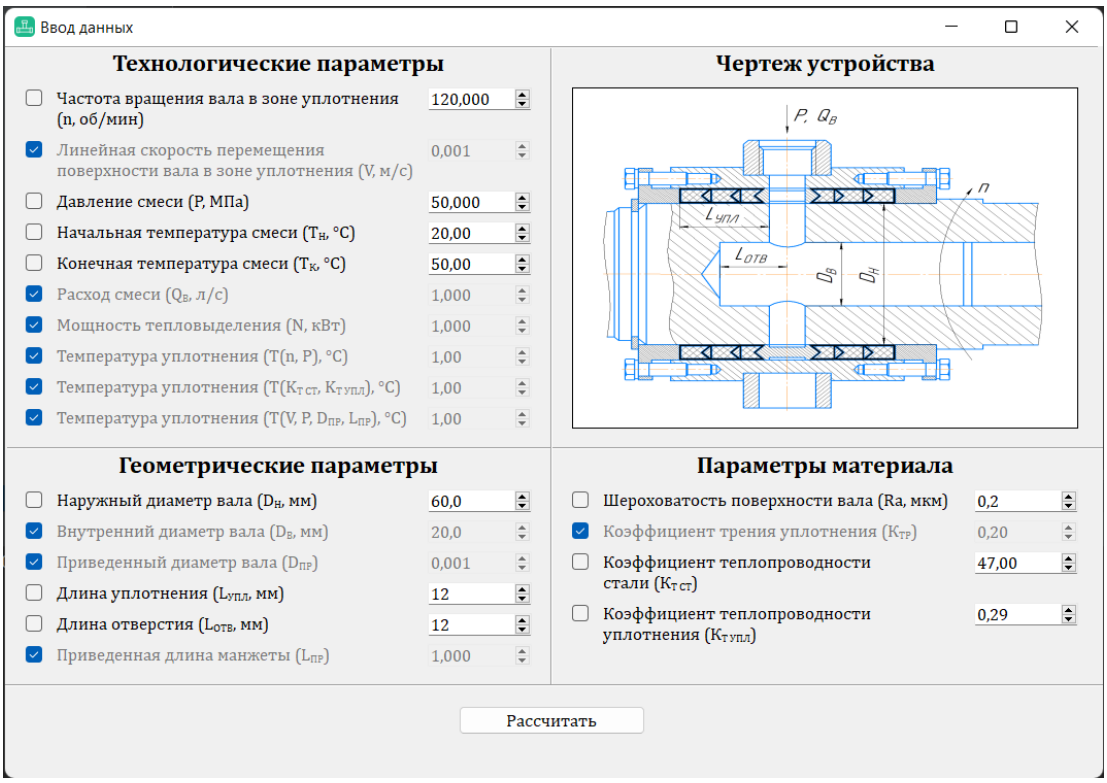

Рисунок 11 – Работа основного окна «Ввод данных»

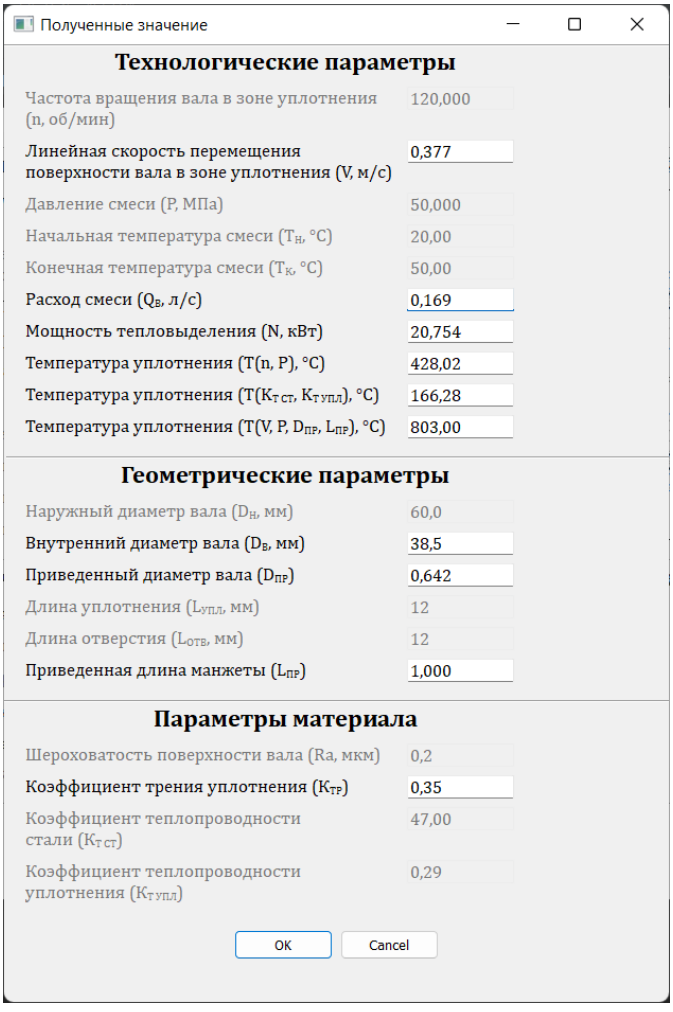

Рисунок 12 – Работа окна результатов «Полученные значения»

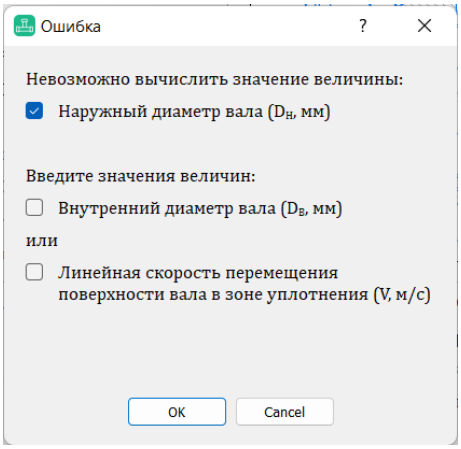

Рисунок 13 – Работа окна сообщения об ошибке «Ошибка»

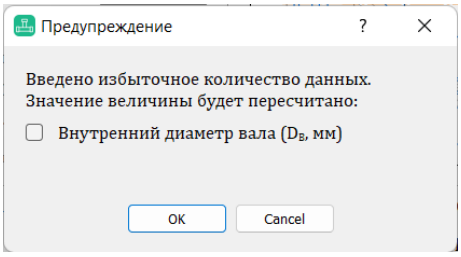

Рисунок 14 – Работа окна предупреждения «Предупреждение»

Также были неоднократно произведены тестирование и отладка программы, в результате которых была достигнута стабильность ее работы.

Подводя итог, нужно сказать, что Qt является обширным и гибким фреймворком, предоставляющим широкий набор инструментов и функциональности для создания приложений с графическим интерфейсом. Он позволяет разработчикам использовать принципы объектно-ориентированного программирования и интегрировать Qt в различные языки программирования.

В дальнейшем планируется доработка и улучшение программы в следующих направлениях: изменение дизайна приложение, которое позволит масштабировать его на экранах с разным разрешением и соотношением сторон; добавление возможности сохранения полученных результатов; добавление возможности обратного порядка расчета и другие.

3. Заключение

Выводы – в результате выполнения курсовой работы по изучению фреймворка Qt и его применения в разработке приложений с графическим интерфейсом были достигнуты следующие результаты:

1. Изучены основные принципы и концепции фреймворка Qt, а также основные компоненты и модули, входящими в состав Qt;
2. Изучены инструменты и средства разработки, предоставляемые Qt, включая Qt Designer, и способы интеграции Qt в различные языки программирования;
3. Разработано приложение, демонстрирующее функциональные возможности Qt, включая создание интерфейса пользователя и обработку событий;
4. Проведено тестирование и отладка разработанного приложения;
5. Сделаны выводы о применимости и эффективности использования Qt в разработке программного обеспечения.

Список литературы

1. Swaroop, C. H. A byte of Python / C. H. Swaroop: 2013. – 189 с.
2. Жирохов, А. Введение в Qt: разработка приложений на C++. / А. Жирохов. – Москва : ДМК Пресс, 2018. – 243 с.
3. Жирохов А. Введение в Qt: разработка приложений на C++. / А. Жирохов. – Москва: ДМК Пресс, 2015. – 167 с.
4. Summerfield M. Программирование на языке Python 3 и PyQt 5. / Summerfield M. – Москва: ДМК Пресс, 2017. – 259 с.
5. Qt Documentation : сайт. – URL: https://doc.qt.io/ (дата обращения: 29.04.2023)

Приложение

```
import math
from PyQt5 import QtCore
from PyQt5 import QtWidgets
from PyQt5.QtGui import QPixmap, QIcon
from PyQt5.QtWidgets import QApplication, QMainWindow, QDialog, QAction
from version2 import Ui_MainWindow
from version21 import Ui_NewWindow
from version22 import Ui_Error
from version23 import Ui_Warning

class MainWindow(QMainWindow, Ui_MainWindow):
    # Конструктор класса
    def __init__(self):
        super().__init__()
        self.setupUi(self)

        # Импрот и установка иконки
        self.setWindowIcon(QIcon('icons8-pipes-96.png'))

        # Импорт и установка чертежа
        self.label_chert.setPixmap(QPixmap("uzel_2.png"))
        self.label_chert.setScaledContents(True)

        # Установка фокуса на первом элементе (значении)
        self.setFocus()
        self.doubleSpinBox_chastota.setFocus()


        # Словарь названий и значений
        self.var_dict = {"chastota": 0, "lin_skoroct": 0, "davlenie": 0, "nach_temp": 0, "konech_temp": 0,
                         "rashod_vody": 0, "moshnoct_tepl": 0, "temp_uplotn_n_P": 0, "temp_uplotn_Kst_Kupl": 0,
                         "temp_uplotn_V_P_Dpr_Lpr": 0, "diam_nar": 0, "diam_vnutr": 0, "diam_priv": 0,
                         "dlina_uplotn": 0, "dlina_otv": 0, "dlina_priv": 0, "sheroh": 0,
                         "tepl_stali": 0, "tepl_uplotn": 0, "trenie_uplotn": 0}

        # Частота и лин. скорость не могут быть выбраны одновременно (но могут быть не выбраны
одновременно)
        self.checkBox_chastota.clicked.connect(lambda:
self.change_with_element(self.checkBox_lin_skoroct))
        self.checkBox_lin_skoroct.clicked.connect(lambda:
self.change_with_element(self.checkBox_chastota))

        # При изменении шероховатости устанавливается флажок в коэфф. трения и наоборот
        self.checkBox_sheroh.clicked.connect(lambda state: self.change_flag(state,
self.checkBox_trenie_uplotn))
        self.checkBox_trenie_uplotn.clicked.connect(lambda state: self.change_flag(state,
self.checkBox_sheroh))
```

```
        # Следующие величины не могут быть отмечены флажками
        self.checkBox_nach_temp.clicked.connect(lambda:
self.checkBox_nach_temp.setCheckState(QtCore.Qt.Unchecked))
        self.checkBox_konech_temp.clicked.connect(lambda:
self.checkBox_konech_temp.setCheckState(QtCore.Qt.Unchecked))
        self.checkBox_dlina_uplotn.clicked.connect(lambda:
self.checkBox_dlina_uplotn.setCheckState(QtCore.Qt.Unchecked))
        self.checkBox_dlina_otv.clicked.connect(lambda:
self.checkBox_dlina_otv.setCheckState(QtCore.Qt.Unchecked))
        self.checkBox_tepl_stali.clicked.connect(lambda:
self.checkBox_tepl_stali.setCheckState(QtCore.Qt.Unchecked))
        self.checkBox_tepl_uplotn.clicked.connect(lambda:
self.checkBox_tepl_uplotn.setCheckState(QtCore.Qt.Unchecked))

        # Следующие величины отмечены флажками, которые нельзя снять
        self.checkBox_rashod_vody.clicked.connect(lambda:
self.checkBox_rashod_vody.setCheckState(QtCore.Qt.Checked))
        self.checkBox_moshnoct_tepl.clicked.connect(lambda:
self.checkBox_moshnoct_tepl.setCheckState(QtCore.Qt.Checked))
        self.checkBox_temp_uplotn_n_P.clicked.connect(lambda:
self.checkBox_temp_uplotn_n_P.setCheckState(QtCore.Qt.Checked))
        self.checkBox_temp_uplotn_Kst_Kupl.clicked.connect(lambda:
self.checkBox_temp_uplotn_Kst_Kupl.setCheckState(QtCore.Qt.Checked))
        self.checkBox_temp_uplotn_V_P_Dpr_Lpr.clicked.connect(lambda:
self.checkBox_temp_uplotn_V_P_Dpr_Lpr.setCheckState(QtCore.Qt.Checked))
        self.checkBox_diam_priv.clicked.connect(lambda:
self.checkBox_diam_priv.setCheckState(QtCore.Qt.Checked))
        self.checkBox_dlina_priv.clicked.connect(lambda:
self.checkBox_dlina_priv.setCheckState(QtCore.Qt.Checked))

        # При установке флажка название и значение становятся не активными, при снятии наоборот
        for name, var in self.var_dict.items():
            self.connect_checkbox(name)

        # Создаем экземпляр нового окна
        self.new_window = NewWindow(self)

        # Создаем экземпляр диалогового окна
        self.error = Error(self)

        # Создаем экземпляр диалогового окна
        self.warning = Warning(self)

        # При нажатии на кнопку запускается функция для расчета
        self.pushButton_calculate.clicked.connect(self.calculate)


    # Функция для снятия флажка с переданного элемента (если на нем стоит флажок, то он снимается)
    def change_with_element(self, element):
```

```
    if element.isChecked():
        element.setChecked(False)

# Функция для смены установки флажков (RadioButton)
def change_flag(self, state, element):
    element.setChecked(not state)

# Функция для связи события и действия по названию
def connect_checkbox(self, name):
    checkbox = getattr(self, f"checkBox_{name}")
    label = getattr(self, f"label_{name}")
    spinbox = getattr(self, f"doubleSpinBox_{name}")
    checkbox.stateChanged.connect(lambda state: self.set_elements_disenabled(state, label, spinbox))

# Функция для смены активности элементов (1 - не акт, 0 - акт)
def set_elements_disenabled(self, state, *elements):
    for element in elements:
        element.setEnabled(not state)

# Функция для получения значений из активных элементов
def get_value_if_element_enabled(self):
    result_dict = {}
    for name, var in self.var_dict.items():
        if getattr(self, f"doubleSpinBox_{name}").isEnabled():
            var = getattr(self, f"doubleSpinBox_{name}").value()
        else:
            var = 0
        result_dict[name] = var
    return result_dict

# Функция для расчета и вывода в новое окно
def calculate(self):

    # Обновление словаря
    updated_dict = self.get_value_if_element_enabled()
    self.var_dict.update(updated_dict)

    # Рассчет параметров переменной активности
    if self.var_dict["davlenie"] != 0:
        t = 4.8 * math.exp(0.0161 * self.var_dict["davlenie"])  # Толщина стенки через давление
        if (self.var_dict["chastota"] != 0 and self.var_dict["lin_skoroct"] == 0) or \
                (self.var_dict["chastota"] == 0 and self.var_dict["lin_skoroct"] != 0):
            if self.var_dict["diam_nar"] != 0:
                if self.var_dict["diam_vnutr"] != 0:
                    self.warning.label_0.setText(self.label_diam_vnutr.text())
                    self.warning.exec_()
                self.var_dict["diam_vnutr"] = self.var_dict["diam_nar"] - 2 * t
            elif self.var_dict["diam_nar"] == 0 and self.var_dict["diam_vnutr"] != 0:
                self.var_dict["diam_nar"] = self.var_dict["diam_vnutr"] + 2 * t
            else:
```

```
                self.error.label_0.setText(self.label_diam_nar.text())
                self.error.label_1.setText(self.label_diam_vnutr.text())
                if self.var_dict["lin_skoroct"] == 0: self.error.label_2.setText(self.label_lin_skoroct.text())
                elif self.var_dict["chastota"] == 0: self.error.label_2.setText(self.label_chastota.text())
                self.error.label_or.setVisible(True)
                self.error.checkBox_2.setVisible(True)
                self.error.exec_()
                return
            if self.var_dict["lin_skoroct"] == 0:
                self.var_dict["lin_skoroct"] = math.pi * self.var_dict["diam_nar"] * self.var_dict["chastota"] /
(60000)
            elif self.var_dict["chastota"] == 0:
                self.var_dict["chastota"] = 60000 * self.var_dict["lin_skoroct"] / (math.pi *
self.var_dict["diam_nar"])
        else:
            if self.var_dict["diam_nar"] == 0 and self.var_dict["diam_vnutr"] == 0:
                self.var_dict["diam_nar"] = 60000 * self.var_dict["lin_skoroct"] / (math.pi *
self.var_dict["chastota"])
            else:
                if self.var_dict["diam_nar"] != 0:
                    self.warning.label_0.setText(self.label_lin_skoroct.text())
                    self.warning.exec_()
                if self.var_dict["diam_vnutr"] != 0:
                    self.warning.label_0.setText(self.label_diam_vnutr.text())
                    self.warning.exec_()
                self.var_dict["lin_skoroct"] = math.pi * self.var_dict["diam_nar"] * self.var_dict["chastota"] /
(60000)
            self.var_dict["diam_vnutr"] = self.var_dict["diam_nar"] - 2 * t
    else:
        if (self.var_dict["chastota"] != 0 and self.var_dict["lin_skoroct"] == 0) or \
                (self.var_dict["chastota"] == 0 and self.var_dict["lin_skoroct"] != 0):
            if self.var_dict["diam_nar"] != 0 and self.var_dict["diam_vnutr"] != 0:
                t = self.var_dict["diam_nar"] - self.var_dict["diam_vnutr"]  # Толщ. стенки ч/з разность диам.
                self.var_dict["davlenie"] = math.log2(t / (2 * 4.8)) / 0.0161
            else:
                if self.var_dict["diam_nar"] == 0:
                    self.error.label_0.setText(self.label_davlenie.text())
                    self.error.label_1.setText(self.label_diam_nar.text())
                    if self.var_dict["lin_skoroct"] == 0: self.error.label_2.setText(self.label_lin_skoroct.text())
                    elif self.var_dict["chastota"] == 0: self.error.label_2.setText(self.label_chastota.text())
                    self.error.label_or.setVisible(True)
                    self.error.checkBox_2.setVisible(True)
                    self.error.exec_()
                    return
                if self.var_dict["diam_vnutr"] == 0:
                    self.error.label_0.setText(self.label_davlenie.text())
                    self.error.label_1.setText(self.label_diam_vnutr.text())
                    self.error.label_2.setText('')
                    self.error.label_or.setVisible(False)
                    self.error.checkBox_2.setVisible(False)
```

```
                self.error.exec_()
                return
        else:
            if self.var_dict["diam_vnutr"] != 0:
                if self.var_dict["diam_nar"] == 0:
                    self.var_dict["diam_nar"] = 60000 * self.var_dict["lin_skoroct"] \
                                / (math.pi * self.var_dict["chastota"])
                else:
                    self.warning.label_0.setText(self.label_lin_skoroct.text())
                    self.warning.exec_()
                    self.var_dict["lin_skoroct"] = math.pi * self.var_dict["diam_nar"] \
                                * self.var_dict["chastota"] / (60000)
                t = self.var_dict["diam_nar"] - self.var_dict["diam_vnutr"]  # Толщ. стенки ч/з разность диам.
                if t > 0: self.var_dict["davlenie"] = math.log2(t / (2 * 4.8)) / 0.0161
                else: self.var_dict["davlenie"] = 0
            else:
                self.error.label_0.setText(self.label_davlenie.text())
                self.error.label_1.setText(self.label_diam_vnutr.text())
                self.error.label_2.setText('')
                self.error.label_or.setVisible(False)
                self.error.checkBox_2.setVisible(False)
                self.error.exec_()
                return
        if self.var_dict["sheroh"] != 0:
            self.var_dict["trenie_uplotn"] = 0.15 * math.log2(self.var_dict["sheroh"]) + 0.7
            print('')
        else:
            self.var_dict["sheroh"] = math.exp((self.var_dict["trenie_uplotn"] - 0.7) / 0.15)
            print('')

        # Рассчет остальных параметров
        self.var_dict["diam_priv"] = self.var_dict["diam_vnutr"] / self.var_dict["diam_nar"]
        self.var_dict["dlina_priv"] = self.var_dict["dlina_otv"] / self.var_dict["dlina_uplotn"]
        self.var_dict["rashod_vody"] = 5 * self.var_dict["trenie_uplotn"] \
                * math.pow(self.var_dict["lin_skoroct"], 1.418) * math.pow(self.var_dict["davlenie"], 0.843) \
                * math.pow(self.var_dict["nach_temp"], 0.253) / math.pow(self.var_dict["konech_temp"],
1.282)
        self.var_dict["moshnoct_tepl"] = 4.25 * math.pow(self.var_dict["lin_skoroct"], 0.576) \
                * math.pow(self.var_dict["trenie_uplotn"], 0.876) * math.pow(self.var_dict["davlenie"],
0.783)
        self.var_dict["temp_uplotn_n_P"] = 3 * math.pow(self.var_dict["chastota"], 0.514) \
                            * math.pow(self.var_dict["davlenie"], 0.639)
        self.var_dict["temp_uplotn_Kst_Kupl"] = 6722 * math.pow(self.var_dict["tepl_uplotn"], 0.0027) \
                            / math.pow(self.var_dict["tepl_stali"], 0.96)
        self.var_dict["temp_uplotn_V_P_Dpr_Lpr"] = \
                130 * math.pow(self.var_dict["davlenie"], 0.59) * math.pow(self.var_dict["lin_skoroct"], 0.445) \
                    * math.pow(self.var_dict["diam_priv"], 0.12) * math.pow(self.var_dict["dlina_priv"], 1.08)

        # Выгрузка значений в новое окно и его вывод
        self.new_window.set_value_and_change_enable(self.var_dict)
```

```
        self.new_window.show()


class NewWindow(QMainWindow, Ui_NewWindow):
    # Конструктор класса
    def __init__(self, main_window):
        super().__init__()
        self.main_window = main_window  # Сохранение ссылки на MainWindow
        self.setupUi(self)

    # Функция для вывода значений в новое окно
    def set_value_and_change_enable(self, values_dict):
        for name, value in values_dict.items():
            label = getattr(self, f"label_{name}")
            doubleSpinBox = getattr(self, f"doubleSpinBox_{name}")
            enable = getattr(self.main_window, f"doubleSpinBox_{name}").isEnabled()
            label.setEnabled(not enable)
            doubleSpinBox.setEnabled(not enable)
            getattr(self, f"doubleSpinBox_{name}").setValue(value)

class Error(QDialog, Ui_Error):
    # Конструктор класса
    def __init__(self, parent=None):
        super(Error, self).__init__(parent)
        self.setupUi(self)

        self.checkBox_2.setVisible(False)
        self.checkBox_3.setVisible(False)
        self.checkBox_0.clicked.connect(lambda: self.checkBox_0.setCheckState(QtCore.Qt.Checked))

class Warning(QDialog, Ui_Warning):
    # Конструктор класса
    def __init__(self, parent=None):
        super(Warning, self).__init__(parent)
        self.setupUi(self)

        #self.buttonBox.button(self.buttonBox.Cancel).clicked.connect(self.reject)

if __name__ == "__main__":
    app = QApplication([])
    window = MainWindow()
    window.show()
    app.exec_()
```