TOMSK POLYTECHNIC UNIVERSITY ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Министерство науки и высшего образования Российской Федерации
федеральное государственное автономное
образовательное учреждение высшего образования
«Национальный исследовательский Томский политехнический университет» (ТПУ)

Школа Инженерная школа неразрушающего контроля и безопасности
Направление подготовки 15.04.01 Машиностроение
ООП/ОПОП Машины и технологии сварочного производства
Специализация Машины и технологии сварочного производства
Отделение Электронной инженерии

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА МАГИСТРАНТА

Тема работы
Разработка технологии захвата, обработки и визуализации электрических параметров режима сварки

УДК 621.791.7.01

Обучающийся

Группа	ФИО	Подпись	Дата
1ВМ11	Иванов Владимир Николаевич		

Руководитель ВКР

Должность	ФИО	Ученая степень, звание	Подпись	Дата
доцент ОЭИ	Гордынец Антон Сергеевич	к.т.н.		

КОНСУЛЬТАНТЫ ПО РАЗДЕЛАМ:

По разделу «Финансовый менеджмент, ресурсоэффективность и ресурсосбережение»

Должность	ФИО	Ученая степень, звание	Подпись	Дата
Профессор ОСГН	Гасанов Магеррам Али оглы	д.э.н., доцент		

По разделу «Социальная ответственность»

Должность	ФИО	Ученая степень, звание	Подпись	Дата
Доцент ООД	Антоневич Ольга Алексеевна	к.б.н.		

Нормоконтроль

Должность	ФИО	Ученая степень, звание	Подпись	Дата
Доцент ОЭИ	Дерюшева Валентина Николаевна	к.т.н.		

ДОПУСТИТЬ К ЗАЩИТЕ:

Руководитель ООП, должность	ФИО	Ученая степень, звание	Подпись	Дата
Доцент ОЭИ	Гордынец Антон Сергеевич	к.т.н.		

Томск – 2023 г.

ПЛАНИРУЕМЫЕ РЕЗУЛЬТАТЫ ОСВОЕНИЯ ООП

Код компетенции	Наименование компетенции
Универсальные компетенции	
УК(У)-1	Способен осуществлять критический анализ проблемных ситуаций на основе системного подхода, вырабатывать стратегию действий
УК(У)-2	Способен управлять проектом на всех этапах его жизненного цикла
УК(У)-3	Способен организовывать и руководить работой команды, вырабатывая командную стратегию для достижения поставленной цели
УК(У)-4	Способен применять современные коммуникативные технологии, в том числе на иностранном(ых) языке(ах), для академического и профессионального взаимодействия
УК(У)-5	Способен анализировать и учитывать разнообразие культур в процессе межкультурного взаимодействия
УК(У)-6	Способен определять и реализовывать приоритеты собственной деятельности и способы ее совершенствования на основе самооценки
Общепрофессиональные компетенции	
ОПК(У)-1	Способностью формулировать цели и задачи исследования, выявлять приоритеты решения задач, выбирать и создавать критерии оценки
ОПК(У)-2	Способностью применять современные методы исследования, оценивать и представлять результаты выполненной работы
ОПК(У)-3	Способностью использовать иностранный язык в профессиональной сфере
ОПК(У)-4	Способностью осуществлять экспертизу технической документации
ОПК(У)-5	Способностью организовывать работу коллективов исполнителей, принимать исполнительские решения в условиях спектра мнений,

	определять порядок выполнения работ, организовывать в подразделении работы по совершенствованию, модернизации, унификации выпускаемых изделий, и их элементов, по разработке проектов стандартов и сертификатов, обеспечивать адаптацию современных версий систем управления качеством к конкретным условиям производства на основе международных стандартов
ОПК(У)-6	Способностью к работе в многонациональных коллективах, в том числе при работе над междисциплинарными и инновационными
ОПК(У)-7	Способностью обеспечивать защиту и оценку стоимости объектов интеллектуальной деятельности
ОПК(У)-8	Способностью проводить маркетинговые исследования и подготавливать бизнес-планы выпуска и реализации перспективных и конкурентоспособных изделий в области машиностроения
ОПК(У)-9	Способностью обеспечивать управление программами освоения новой продукции и технологий, проводить оценку производственных и непроизводственных затрат на обеспечение требуемого качества продукции, анализировать результаты деятельности производственных подразделений
ОПК(У)-10	Способностью организовывать работу по повышению научно-технических знаний работников
ОПК(У)-11	Способностью подготавливать отзывы и заключения на проекты стандартов, рационализаторские предложения и изобретения в области машиностроения
ОПК(У)-12	Способностью подготавливать научно-технические отчеты, обзоры, публикации по результатам выполненных исследований в области машиностроения
ОПК(У)-13	Способностью разрабатывать методические и нормативные документы, предложения и проводить мероприятия по реализации разработанных проектов и программ в области машиностроения
ОПК(У)-14	Способностью выбирать аналитические и численные методы при разработке математических моделей машин, приводов, оборудования, систем, технологических процессов в машиностроении
Профессиональные компетенции	
ПК(У)-1	Способностью разрабатывать технические задания на проектирование и изготовление машин, приводов, оборудования, систем и нестандартного оборудования и средств технологического оснащения, выбирать оборудование и технологическую оснастку

ПК(У)-2	Способностью разрабатывать нормы выработки и технологические нормативы на расход материалов, заготовок, топлива и электроэнергии в машиностроении
ПК(У)-3	Способностью оценивать технико-экономическую эффективность проектирования, исследования, изготовления машин, приводов, оборудования, систем, технологических процессов, принимать участие в создании системы менеджмента качества на предприятии
ПК(У)-8	Способностью организовать и проводить научные исследования, связанные с разработкой проектов и программ, проводить работы по стандартизации технических средств, систем, процессов оборудования и материалов
ПК(У)-9	Способностью разрабатывать физические и математические модели исследуемых машин, приводов, систем, процессов, явлений и объектов, относящихся к профессиональной сфере, разрабатывать методики и организовывать проведение экспериментов с анализом их результатов
ПК(У)-10	Способностью и готовностью использовать современные психолого-педагогические теории и методы в профессиональной деятельности

TOMSK POLYTECHNIC UNIVERSITY ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Министерство науки и высшего образования Российской Федерации
федеральное государственное автономное
образовательное учреждение высшего образования
«Национальный исследовательский Томский политехнический университет» (ТПУ)

Школа	Инженерная школа неразрушающего контроля и безопасности
Направление подготовки	15.04.01 Машиностроение
ООП/ОПОП	Машины и технологии сварочного производства
Специализация	Машины и технологии сварочного производства
Отделение	электронной инженерии

УТВЕРЖДАЮ:
Руководитель ООП
_____ _____ А.С. Гордынец
(Подпись) (Дата) (ФИО)

ЗАДАНИЕ
на выполнение выпускной квалификационной работы

Обучающийся:

Группа	ФИО
1ВМ11	Иванов Владимир Николаевич

Тема работы:

Разработка технологии захвата, обработки и визуализации электрических параметров режима сварки	
Утверждена приказом директора	*№ 39-35/с от 08.02.2023*

Срок сдачи обучающимся выполненной работы:	

ТЕХНИЧЕСКОЕ ЗАДАНИЕ:

Исходные данные к работе *(наименование объекта исследования или проектирования; производительность или нагрузка; режим работы (непрерывный, периодический, циклический и т. д.); вид сырья или материал изделия; требования к продукту, изделию или процессу; особые требования к функционированию (эксплуатации) объекта или изделия в плане безопасности эксплуатации, влияния на окружающую среду, энергозатратам; экономический анализ и т. д.)*	Программа; язык программирования; фреймворк; сварочный процесс; осциллограф; перечень существующей документации к языку программирования и фреймворку
Перечень разделов пояснительной записки подлежащих исследованию, проектированию и разработке *(аналитический обзор литературных источников с целью выяснения достижений мировой науки техники в рассматриваемой области; постановка задачи исследования, проектирования, конструирования; содержание процедуры исследования, проектирования, конструирования; обсуждение результатов выполненной работы; наименование дополнительных разделов, подлежащих разработке; заключение по работе)*	Введение 1 Обзор литературы 2 Разработка прогаммы 3 Финансовый менеджмент, ресурсоэффективность и ресурсосбережение 4 Социальная ответственность Заключение
Перечень графического материала *(с точным указанием обязательных чертежей)*	Титульный лист Название темы, цель, задачи Код программы
Консультанты по разделам выпускной квалификационной работы	

(с указанием разделов)	
Раздел	**Консультант**
Финансовый менеджмент, ресурсоэффективность и ресурсосбережение	Гасанов Магеррам Али оглы, д.э.н., профессор ОСГН
Социальная ответственность	Антоневич Ольга Алексеевна, к.б.н., доцент ООД
Английский язык	Щеголихина Юлия Викторовна, к.ф.н., доцент ОИЯ
Названия разделов, которые должны быть написаны на иностранном языке:	
Введение	
Обзор литературы	
Платформа и язык программирования	
Платформа .Net	
Язык программирования C#	
Инструмент для создания графического интерфейса Windows Forms	
Подходы и паттерны, применимые в разработке	
Dependency Injection	
Паттерн MVVM	
Алгоритмы	
Временная сложность алгоритма	

Дата выдачи задания на выполнение выпускной квалификационной работы по линейному графику	

Задание выдал руководитель:

Должность	ФИО	Ученая степень, звание	Подпись	Дата
Доцент ОЭИ	Гордынец Антон Сергеевич	к.т.н.		

Задание принял к исполнению обучающийся:

Группа	ФИО	Подпись	Дата
1ВМ11	Иванов Владимир Николаевич		

TOMSK POLYTECHNIC UNIVERSITY ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Министерство науки и высшего образования Российской Федерации
федеральное государственное автономное
образовательное учреждение высшего образования
«Национальный исследовательский Томский политехнический университет» (ТПУ)

Школа Инженерная школа неразрушающего контроля и безопасности
Направление подготовки 15.04.01 Машиностроение
ООП/ОПОП Машины и технологии сварочного производства
Специализация Машины и технологии сварочного производства
Отделение электронной инженерии

КАЛЕНДАРНЫЙ РЕЙТИНГ-ПЛАН
выполнения выпускной квалификационной работы

Обучающийся:

Группа	ФИО
1ВМ11	Иванов Владимир Николаевич

Тема работы:

Разработка технологии захвата, обработки и визуализации электрических параметров режима сварки

Срок сдачи обучающимся выполненной работы:	

Дата контроля	Название раздела (модуля) / вид работы (исследования)	Максимальный балл раздела (модуля)
17.02.2023	Обзор литературы	20
28.04.2023	Разработка программы	50
05.05.2023	Финансовый менеджмент, ресурсоэффективность и ресурсосбережение	10
15.05.2023	Социальная ответственность	10
01.06.2023	Заключение	10

СОСТАВИЛ:
Руководитель ВКР

Должность	ФИО	Ученая степень, звание	Подпись	Дата
Доцент ОЭИ	Гордынец А.С.	к.т.н.		

СОГЛАСОВАНО:
Руководитель ООП

Должность	ФИО	Ученая степень, звание	Подпись	Дата
Доцент ОЭИ	Гордынец А.С.	к.т.н.		

Обучающийся

Группа	ФИО	Подпись	Дата
1ВМ11	Иванов Владимир Николаевич		

РЕФЕРАТ

Выпускная квалификационная работа 121 с., 32 рис., 17 табл., 36 источников, 1 прил.

Ключевые слова: технология захвата, обработки и визуализации электрических параметров режима сварки, разработка программного обеспечения.

Объектом разработки является технология захвата, обработки и визуализации электрических параметров режима сварки

Цель работы – разработать прототип программного обеспечения для захвата, обработки и визуализации электрических параметров сварочного процесса.

В ходе работы проводилось изучение инструментов и подходов к разработке программного обеспечения, способы взаимодействия с электронным испытательным оборудованием, разработка прототипа программы для захвата, обработки и визуализации электрических параметров сварочного процесса, а также испытания разработанного программного обеспечения.

В результате был получен исправно работающий прототип представленной системы.

Степень внедрения: прототип.

Область применения: разработанная система в будущем может применяться научными институтами, а также частными компаниями, занимающимися разработкой сварочного оборудования или новых методик сварки в качестве основного или дополнительного инструмента для анализа процесса сварки. Результаты работы позволят автоматизировать процесс регистрации, обработки и хранения данных, содержащих информацию о характере протекания процессов сварки. Такая технология исключит субъективный фактор и поспособствует дальнейшему развитию сварочного производства.

Экономическая эффективность работы обусловлена отсутствием прямых конкурентов в сфере анализа и наличием потребности в более глубоком изучении сварочных процессов.

В будущем планируется доработка представленного прототипа, до полноценной программы, пригодной для использования научными сотрудниками.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Строительство, автомобильная и авиационная промышленность, и многие другие не обходятся без сварки металлов. В настоящий момент сварка – один из важнейших этапов производства. В зависимости от производственных условий применяют самые разные методы сварки: от сварки покрытым электродом или трением, до электронно-лучевой или взрывом. Сварочный процесс надежно закрепился на производстве и с развитием оборудования, развитие требуется самому процессу сварки

В настоящее время многие компании, как отечественные, так и зарубежные, производящие сварочное оборудование, занимаются совершенствованием сварочного процесса. Разрабатываются «новые» способы сварки, зачастую лишь незначительно отличающиеся друг от друга. Для того, чтобы достигнуть прогресса в этом направлении, необходимо начать проводить более детальный анализ сварочного процесса.

В данной выпускной квалификационной работе для осуществления первого шага в развитии инструментов анализа данных сварочного процесса предлагается разработанный прототип программного обеспечения, которое совместно с регистрирующим оборудованием (осциллографом) позволит более детально увидеть, а также проанализировать поступающие данные о процессе сварки.

В связи с изложенным целью данной работы является: выбор инструмента для прототипирования программного обеспечения, анализ имеющейся литературы о работе с выбранным инструментом, разработка прототипа с последующим проведением тестов.

Для достижения поставленной цели необходимо решить следующие задачи: выбрать инструмент для прототипирования приложения, разработать модули чтения, обработки, хранения и визуализации снимаемых данных, разработать модуль для подключения сторонних инструментов анализа данных, провести тестирование прототипа.

1 ОБЗОР ЛИТЕРАТУРЫ

Для того, чтобы начать прототипирование проекта необходимо определиться с тем, какие инструменты будут использоваться в ходе разработки: языком программирования, фреймворком и библиотекой, паттернами, облегчающими задачу и т.д.

В данном разделе будут представлены выбранные инструменты, используемые при выполнении работ, а также дана вводная информация по алгоритмам.

1.1 Платформа и язык программирования

При выборе платформы и языка программирования для реализации своего проекта необходимо учитывать его цели, требования к производительности, масштабируемости и безопасности, а также возможности интеграции с другими технологиями. Платформа .NET и язык программирования C# являются мощными инструментами для создания приложений и имеют большое количество преимуществ, которые могут быть использованы в различных областях, от разработки десктопных приложений до создания веб-сервисов и мобильных приложений.

1.1.1 Платформа .Net

.NET Framework – впервые выпущенная вместе с языком программирования C# в 2002 году платформа от Microsoft, предназначенная для разработки программного обеспечения. Во многом похожая на Java, долгое время считалась её прямым конкурентом и альтернативой. Главным отличием от продукта компании Sun является направленность на работу в первую очередь с операционными Microsoft Windows. На сегодняшний день Microsoft прекратила ее развитие, сконцентрировавшись на новом поколении – .Net Core, но несмотря на долгое отсутствие обновлений имеет большую популярность: было создано внушительное количество продуктов, а обширное

и лояльное сообщество по-прежнему поддерживает свои проекты разработанные под .Net Framework.

На замену .Net Framework в 2016 году пришла .Net Core (сейчас переименованная в просто .Net), имеющая в своей основе идею кроссплатформенности, что позволяет создавать на ней проекты под различные операционные системы и оборудование: от smart-телевизоров до стиральных машин. С тех пор платформа от Microsoft стала более привлекательной для бизнеса, поскольку позволяла покрывать ещё больший спектр задач. [1]

Кроссплатформенность современной .Net возможна благодаря переводу кода приложений, написанных на одном из языков платформы, в код на промежуточном языке (Intermediate Language, IL) и сохраняет его в сборке (DLL- или EXE-файле).

Операторы кода на промежуточном языке (IL) похожи на код ассемблера, только выполняются с помощью виртуальной машины CoreCLR в .NET. В процессе работы код IL загружается CoreCLR из сборки, динамически (just-intime, JIT) компилируется компилятором в собственные инструкции CPU, а затем исполняется с помощью CPU на вашем компьютере.

Преимущество такого трехэтапного процесса компиляции заключается в том, что Microsoft может создавать CLR не только для Windows, но и для Linux и macOS. Один и тот же код IL запускается в любой среде благодаря второму процессу компиляции, который генерирует код для конкретной операционной системы и набора команд CPU. [2]

.Net Core предлагает разработчикам удобную платформу для создания приложений рабочего стола на основе Windows Forms, WPF или UWP, веб-приложений, служб, библиотек и многого другого. Также обновленная .Net Core предоставляет возможность двустороннего взаимодействия между ней и .Net Framework за счет системы .Net Standard. .Net поддерживает несколько языков программирования (C#, F#, а также VB.Net). Библиотека базовых классов .Net предоставляет множество готовых классов для построения

приложений. Библиотеки .NET Core не регистрируются в системном реестре. Более того, платформа .NET Core позволяет нескольким версиям инфраструктуры и приложения гармонично сосуществовать на одном компьютере. Интерфейс командной строки .NET Core (command-line interface — CLI) является межплатформенной цепочкой инструментов для разработки и пакетирования приложений .NET Core. Помимо стандартных инструментов, поставляемых в составе .NET Core SDK, могут быть установлены дополнительные инструменты. [3]

.Net Core имеет три ключевых (и взаимосвязанных) компонента, которые делают возможным все упомянутое ранее — Core Runtime (формально CoreCLR и CoreFX), CTS и CLS. С точки зрения разработчика приложений платформу .NET Core можно воспринимать как исполняющую среду и обширную библиотеку базовых классов. Уровень исполняющей среды содержит набор минимальных реализаций, которые привязаны к конкретным платформам (Windows, iOS, Linux) и архитектурам (x86, x64, ARM), а также все базовые типы для .NET Core.

Одним из столпов платформы .Net является CTS – общая система типов – в ее спецификации описаны все типы данных и программные конструкции, которые поддерживает среда выполнения. Она описывает способы взаимодействия типов и программных конструкций между собой, также их запись в форме метаданных.

Необходимо понимать, что не все функциональные возможности, определенные в спецификации CTS, могут быть поддержаны отдельно взятым языком .NET Core. Однако существует общеязыковая спецификация CLS, которая описывает подмножество типов и программных конструкций, которые должны быть поддержаны всеми языками программирования .NET Core. Если вы создаете типы .NET Core, которые используют только совместимые с CLS средства, то можно быть уверенным, что их смогут использовать все языки .NET Core. Но если вы используете тип данных или программную конструкцию, которая не поддерживается CLS, то нельзя

гарантировать, что каждый язык программирования .NET Core сможет использовать вашу библиотеку кода .NET Core.

Набор библиотек базовых классов (BCL), предоставляемых инфраструктурой .NET Core, доступен всем языкам программирования .NET Core и включает в себя множество примитивов, таких как потоки, файловый ввод-вывод, системы визуализации графики и взаимодействие с внешними устройствами, а также поддержку для различных служб, необходимых для большинства приложений. Библиотеки базовых классов содержат типы, которые могут использоваться для создания любого типа приложения и компонентов, взаимодействующих друг с другом.

Даже с учетом выхода .NET 7.0 количество библиотек базовых классов в .NET Framework намного превышает количество библиотек подобного рода в .NET Core. Учитывая 14-летнее преимущество .NET Framework над .NET Core, ситуация вполне объяснима. Такое несоответствие создает проблемы при попытке использования кода .NET Framework с кодом .NET Core. Решением (и требованием) для взаимодействия .NET Framework/.NET Core является стандарт .NET Standard.

.NET Standard — это спецификация, определяющая доступность API-интерфейсов .NET и библиотек базовых классов, которые должны присутствовать в каждой реализации. Стандарт обладает следующими характеристиками:

– определяет унифицированный набор API-интерфейсов BCL для всех реализаций .NET, которые должны быть созданы независимо от рабочей нагрузки;

– позволяет разработчикам производить переносимые библиотеки, пригодные для потребления во всех реализациях .NET, с использованием одного и того же набора API-интерфейсов;

– сокращает или даже устраняет условную компиляцию общего исходного кода API-интерфейсов .NET, оставляя ее только для API-интерфейсов операционной системы.

В таблице, приведенной в документации от Microsoft (https://docs.microsoft.com/ru-ru/dotnet/standard/net-standard), указаны минимальные версии реализаций, которые поддерживают каждый стандарт .NET Standard. [3]

1.1.2 Язык программирования C#

Хотя синтаксис языка C# похож на синтаксис языка Java, нельзя назвать его клоном Java, так как оба языка принадлежат к семейству языков программирования, основанных на C. Кроме того, C# заимствует конструкции из языков VB и C++, а также функциональных языков программирования, таких как LISP и Haskell. Однако наибольшее влияние на C# оказали именно языки, основанные на C. В C# есть возможность использовать понятия свойств класса, необязательных параметров, перегрузки операций, создания структур, перечислений и функций обратного вызова. Кроме того, C# поддерживает лямбда-выражения и анонимные типы, которые встречаются в функциональных языках программирования. Технология LINQ делает язык C# уникальным в мире программирования. Кроме того, в языке отсутствует необходимость в прямых манипуляциях указателями и есть автоматическое управление памятью через сборку мусора. Также имеются формальные синтаксические конструкции для классов, интерфейсов, структур, перечислений и делегатов, а также поддержка программирования на основе атрибутов.

C# 9 уже является мощным языком, который в сочетании с .NET Core позволяет строить широкий спектр приложений разнообразных видов.

C# является языком программирования, который может использоваться только для создания программного обеспечения, работающего на исполняющей среде .NET Core. Это означает, что C# не может использоваться для создания COM-серверов или неуправляемых приложений в стиле C/C++. Код, написанный на C# и ориентированный на исполняющую среду .NET Core, называется управляемым кодом. Такой код содержится в двоичном

модуле, который называется сборкой. Если код не может напрямую обслуживаться управляющей средой платформы, то его называют неуправляемым кодом.

Как упоминалось ранее, инфраструктура .NET Core способна функционировать в средах разнообразных операционных систем. Таким образом, вполне вероятно создавать приложение C# на машине Windows с применением Visual Studio и запускать его под управлением iOS с использованием исполняющей среды .NET Core. Кроме того, приложение C# можно построить на машине Linux с помощью Visual Studio Code и запускать его на машине Windows. С помощью Visual Studio для Mac на компьютере Mac можно разрабатывать приложения .NET Core, предназначенные для выполнения под управлением Windows, macOS или Linux.

Программа C# по-прежнему может иметь доступ к неуправляемому коду, но тогда она привяжет вас к специфической цели разработки и развертывания. [3]

1.2 Инструмент для создания графического интерфейса Windows Forms

Windows Forms – это платформа пользовательского интерфейса, которая обеспечивает эффективное создание классических приложений Windows с помощью визуального конструктора в Visual Studio. Она представляет собой набор управляемых библиотек для выполнения стандартных задач, таких как чтение и запись в файловую систему. Приложения Windows Forms могут быть графически сложными и легко развертываться и обновляться. Форма в Windows Forms – это визуальная поверхность, на которой отображается информация для пользователя, а элементы управления предназначены для ввода и отображения данных. В Windows Forms имеется множество элементов управления, а также функции перетаскивания и выравнивания для быстрого создания сложных макетов форм. Пространство имен System.Drawing содержит классы для отрисовки фигур на форме. [4]

Windows Forms обеспечивает простой способ отображения данных из различных источников, таких как базы данных, файлы XML или JSON, веб-службы и другие. Элемент управления DataGridView позволяет отображать табличные данные в традиционном формате, где каждый фрагмент данных занимает свою собственную ячейку. Кроме того, DataGridView позволяет настроить внешний вид ячеек и зафиксировать строки и столбцы на своем месте.

Windows Forms также обеспечивает простой способ подключения к источникам данных по сети. Компонент BindingSource представляет подключение к источнику данных и содержит методы для привязки данных к элементам управления, перехода между записями, редактирования записей и сохранения изменений в исходном источнике. Элемент управления BindingNavigator предоставляет простой интерфейс для перехода между записями.

С помощью окна "Источники данных" в Visual Studio можно легко создавать элементы управления с привязкой к данным. В этом окне отображаются существующие в проекте источники данных, такие как базы данных, веб-службы и объекты. Создавать элементы управления с привязкой к данным можно путем перетаскивания объектов из этого окна в формы проекта. Также можно связывать существующие элементы управления с данными, перетаскивая объекты из окна "Источники данных" в существующие элементы управления.

Windows Forms также предоставляет возможность сохранять некоторые сведения о состоянии приложения во время выполнения, например, последний известный размер форм и пользовательские предпочтения. Это можно сделать с помощью параметров приложения, которые предоставляют простой способ хранения этих сведений на клиентском компьютере в XML-файле. Параметры приложения определяются с помощью Visual Studio или редактора кода и автоматически считываются обратно в память во время выполнения.

1.3 Подходы и паттерны, применимые в разработке

Многие задачи были решены уже множество раз, для каких-то со временем были разработаны более эффективные подходы и модели поведения.

В данном разделе будет уделено внимание применяемым при реализации проекта подходам и паттернам программирования, будет дана общая информация о них, а позднее, в разделе посвященном ходу работ будет рассказано о том, что дало использование этих шаблонов.

1.3.1 Dependency Injection

Dependency Injection или внедрение зависимостей представляет собой набор принципов и приемов проектирования программных продуктов, позволяющий разрабатывать слабосвязанный код.

Эта технология является не столько самоцелью, сколько средством достижения результата. В конечном счете предназначением большинства методов программирования является предоставление работоспособного программного продукта наиболее эффективным способом. При этом один из аспектов — написание сопровождаемого кода.

Довольно быстро обнаружится, что существующий код нуждается в расширении и сопровождении. В целом эффективность работы с таким кодом определяется степенью его сопровождаемости. [5]

Отличным способом повышения сопровождаемости является ослабление связывания. Это было известно еще в 1994 году, когда «Банда четырех» работала над книгой «Паттерны проектирования»: «программируйте в соответствии с интерфейсом, а не с реализацией». [6]

Слабое связывание придает коду расширяемость, а та, в свою очередь, делает его сопровождаемым. DI — не что иное, как технология, обеспечивающая слабое связывание.

Использование DI является не конечной целью, а средством достижения определенного результата. Технология DI позволяет применять слабое связывание, которое, в свою очередь, делает код сопровождаемым. [5]

1.3.2 Паттерн MVVM

Архитектурный паттерн MVVM (Model-View-ViewModel) позволяет отделить внутреннюю логику программного обеспечения от его интерфейса. MVVM состоит из трех компонентов: модели (Model), модели представления (ViewModel) и представления (View).

Модель является объектным представлением данных и может содержать логику, связанную с этими данными. Часто модель реализует интерфейсы INotifyPropertyChanged или INotifyCollectionChanged для уведомления системы об изменениях свойств модели. Это упрощает привязку к представлению, но прямое взаимодействие между моделью и представлением отсутствует. [7]

Представление — это пользовательский интерфейс приложения, через который пользователь взаимодействует с приложением.

Любые интеллектуальные возможности необходимо встраивать в какие-то другие места приложения. Иметь прямое отношение к манипулированию пользовательским интерфейсом может только код в файле отделенного кода. Он не должен быть основан на бизнес-правилах или на чем-то еще, что нуждается в предохранении для будущего применения. Хотя это не является главной целью MWM, хорошо разработанные приложения MVVM обычно имеют совсем небольшой объем отделенного кода.

Представление не обрабатывает события за редким исключением, а выполняет действия в основном посредством команд.

Модель представления служит двум целям:

1. Модель представления предлагает единственное местоположение для всех данных, необходимых представлению. Это вовсе не означает, что модель представления отвечает за получение действительных данных; взамен она является просто транспортным механизмом для перемещения данных из хранилища в представление. Обычно между представлениями и моделями представлений имеется отношение “один к одному”, но существуют

архитектурные отличия, которые в каждом конкретном случае могут варьироваться.

2. Вторая цель модели представления касается ее действия в качестве контроллера для представления. Модель представления принимает указание от пользователя и передает их соответствующему коду для выполнения подходящих действий. Довольно часто такой код имеет форму специальных команд.

Преимуществом использования данного паттерна является меньшая связанность между компонентами и разделение ответственности между ними. То есть Model отвечает за данные, View отвечает за графический интерфейс, а ViewModel - за логику приложения.

На рисунке 1 представлена схема взаимодействия компонентов между собой.

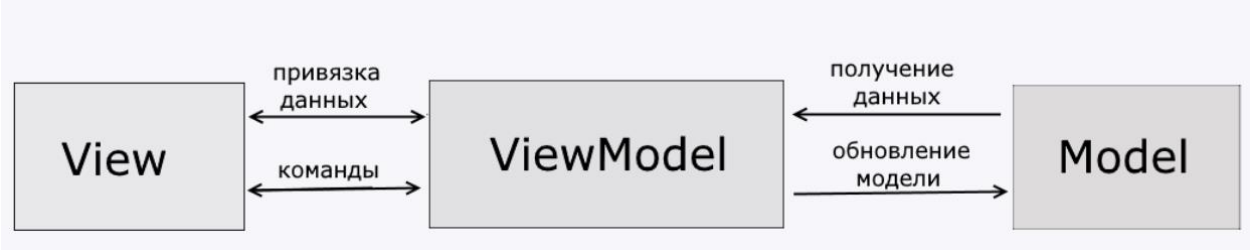


Рисунок 1 – Схема взаимодействия компонентов

Итогом применения паттерна MVVM является функциональное разделение приложения на три компонента, которые проще разрабатывать и тестировать, а также в дальнейшем модифицировать и поддерживать. [3, 7]

1.4 Алгоритмы

Алгоритм — это последовательность шагов, которая решает задачу.

Несмотря на то, что алгоритмы являются фундаментальным понятием в информатике, ученые пока не пришли к единому определению. Существует несколько конкурирующих между собой формулировок, однако предложенная Дональдом Эрвином Кнутом — одна из наиболее известных. Он описывает алгоритм как определенный, эффективный и конечный процесс,

который получает входные данные и производит выходные данные на основе полученных входных.

Определенность означает, что шаги четкие, лаконичные и недвусмысленные.

Эффективность означает, что вы можете выполнить каждую операцию точно для решения задачи.

Конечность подразумевает, что алгоритм останавливается после определенного количества шагов.

Общепринятое дополнение к этому списку — правильность. Алгоритм должен всегда выдавать один и тот же результат для заданных входных данных, и полученный результат должен быть правильным ответом на задачу, которую решает алгоритм.

Не все, но большая часть алгоритмов удовлетворяют этим требованиям, однако некоторые исключения важны. Скажем, когда вы создаете генератор случайных чисел, ваша цель — сгенерировать случайность, поэтому нельзя использовать вывод, чтобы понять, каким был ввод. Кроме того, многие алгоритмы в data science не так строго придерживаются правильности. Например, алгоритму может быть достаточно лишь предположить вывод, если заранее известна его неопределенность. Однако в большинстве случаев ваш алгоритм должен соответствовать всем заявленным требованиям.

1.4.1 Временная сложность алгоритма

Одним из основных критериев оценки алгоритма является его скорость решения задачи (временная эффективность), но также важна и сложность его реализации. Не всегда быстрый, но сложный алгоритм является предпочтительным, так как менее сложный алгоритм может упростить процесс отладки. Реализация некоторых алгоритмов может требовать дополнительного объема памяти для хранения промежуточных данных. [8, 9]

Для анализа алгоритмов используются теоретические методы, главным из которых является метод асимптотической оценки временной

эффективности. Он заключается в установлении функциональной зависимости числа действий, выполняемых алгоритмом, от объема исходных данных в предельном случае больших объемов этих данных. При анализе алгоритмов оценку их эффективности делают, исходя из предположения о наихудшем сочетании значений исходных данных с точки зрения времени выполнения. Временная эффективность алгоритмов может выражаться полиномами второй и более высоких степеней от размерности задачи. Однако при переходе к пределу больших n отбрасываются все члены полинома, кроме старшего, что позволяет определить скорость роста для алгоритма – закон изменения времени выполнения от размерности задачи.

Поскольку важной частью алгоритма является та часть, которая растет быстрее по мере увеличения n, для описания эффективности алгоритма программисты вместо равенства T(n) используют нотацию «О большое». Нотация «О большое» — это математическое обозначение, описывающее, как по мере роста n возрастают требования алгоритма к времени и объему памяти.

Программисты используют нотацию «О большое» для создания функции порядка величины T(n). Порядок величины — тип объекта в системе классификации, в которой каждый тип во много раз больше или меньше предыдущего. В функции порядка величины вы используете ту часть T(n), которая преобладает в уравнении, и игнорируете все остальное. Часть T(n), преобладающая в уравнении, — это порядок величины алгоритма. Ниже представлены общепринятые классификации порядка величины для нотации «О большое»: от наилучшего (наиболее эффективного) до наихудшего (наименее эффективного):

1. Постоянное время.
2. Логарифмическое время.
3. Линейное время.
4. Линейно-логарифмическое время.
5. Квадратичное время.
6. Кубическое время.

7. Экспоненциальное время.

Каждый порядок величины описывает временную сложность алгоритма.

Временная сложность — это максимальное количество шагов, требующихся алгоритму для завершения по мере увеличения n. [9]

Рассмотрим подробнее каждый порядок величины.

Постоянное время

Наиболее эффективным порядком величины является постоянная временная сложность. Алгоритм выполняется за постоянное время, когда ему требуется одно и то же количество шагов вне зависимости от объема задачи. Нотация «О большое» для постоянной сложности — O(1).

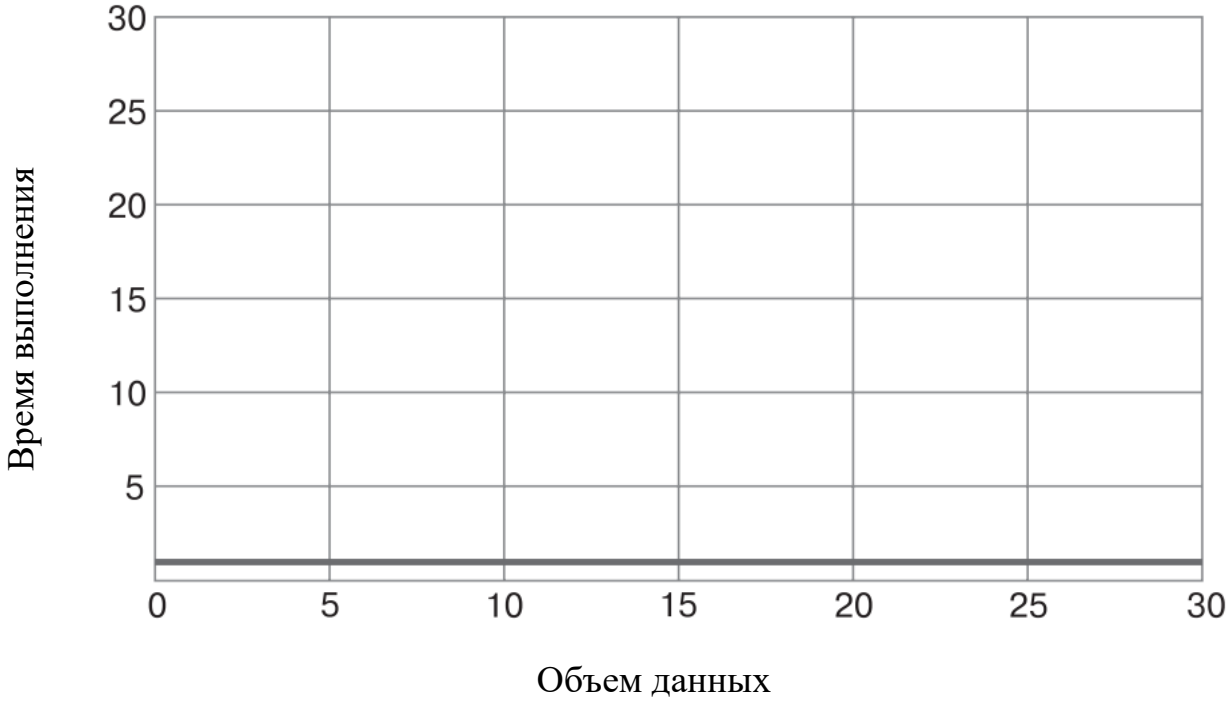


Рисунок 2 – Постоянная временная сложность [9]

Как вы можете видеть, количество шагов, необходимых вашему алгоритму для завершения, не увеличивается по мере роста объема задачи. Таким образом, это и есть наиболее эффективный алгоритм, который можно написать, потому что его время не меняется с увеличением набора данных.

Логарифмическое время

Логарифмическое время — вторая по эффективности временная сложность. Алгоритм выполняется за логарифмическое время, когда время выполнения программы растет пропорционально логарифму размера входных данных. Подобную временную сложность можно увидеть в таких алгоритмах, как двоичный поиск, который может не учитывать множество значений в каждом цикле. (Если сейчас вам что-то непонятно, не переживайте — позднее мы обсудим это более подробно.) Логарифмический алгоритм в нотации «О большое» выражается как O(log n).

Рисунок 3 показывает, как на графике выглядит логарифмический алгоритм.

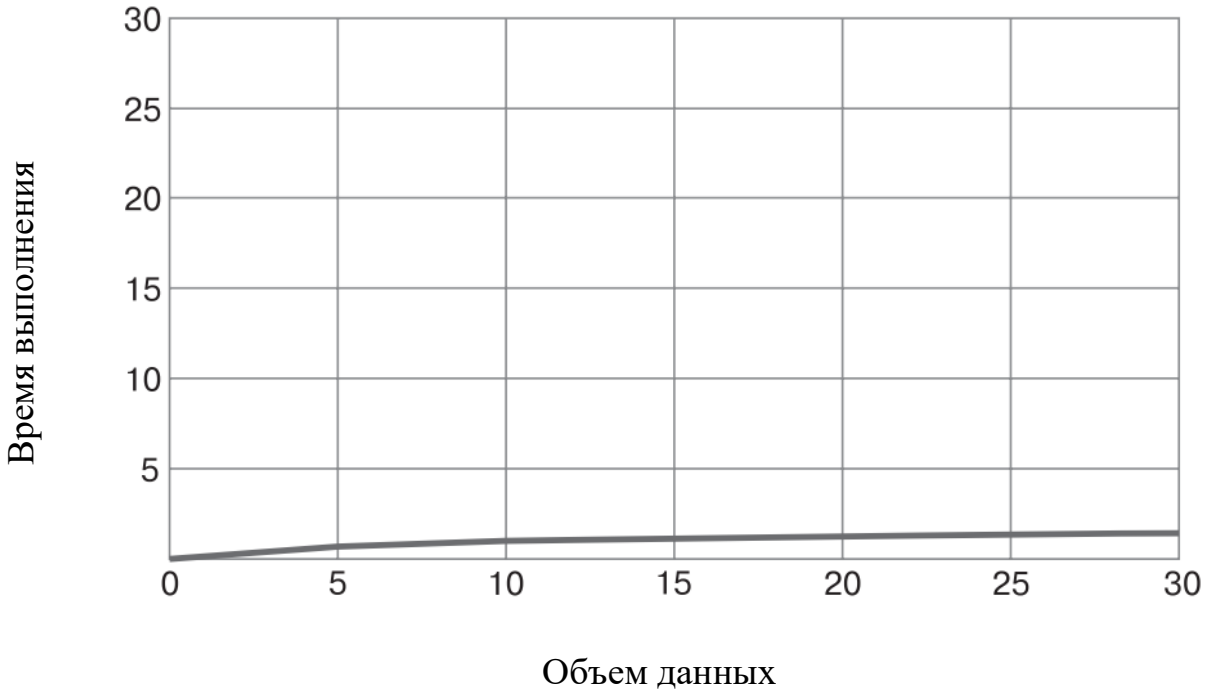


Рисунок 3 – Логарифмическая временная сложность [9]

В логарифмическом алгоритме количество необходимых шагов по мере увеличения набора данных растет достаточно медленно.

Линейное время

Следующий по эффективности тип алгоритма — тот, который выполняется за линейное время. Такой алгоритм растет пропорционально

росту размера задачи. Линейный алгоритм в нотации «О большое» выражается как O(n).

В линейном алгоритме по мере роста n количество шагов алгоритма увеличивается на то же количество, что и n (рисунок 4).

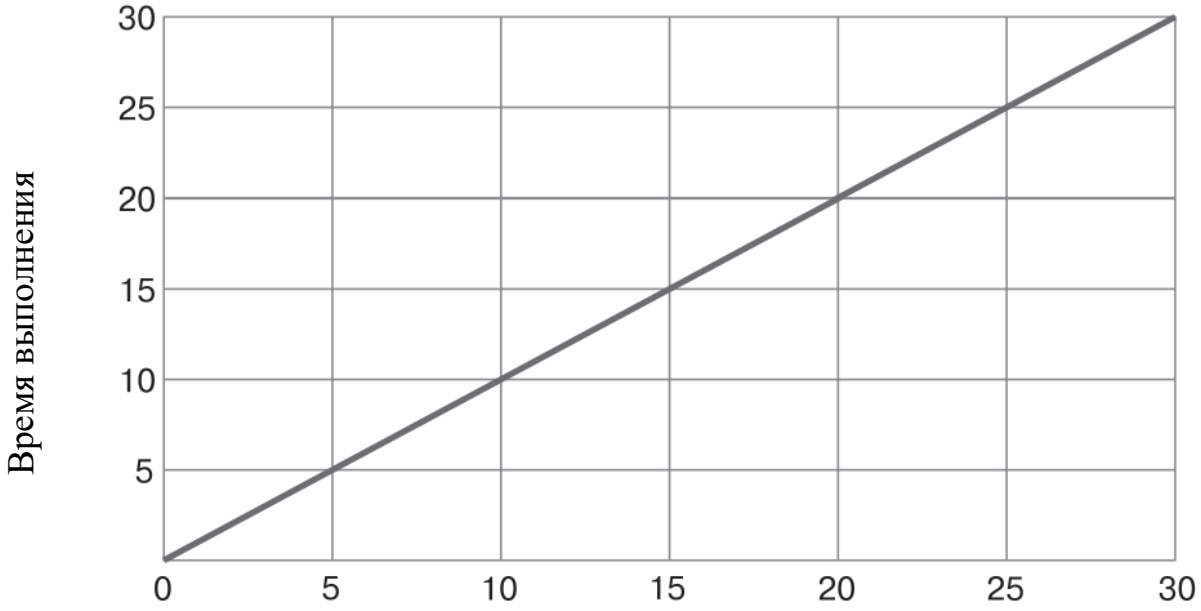


Рисунок 4 – Линейная временная сложность [9]

Линейно-логарифмическое время

Алгоритм, выполняющийся за линейно-логарифмическое время, растет как сочетание (умножение) логарифмических и линейных временных сложностей. Например, линейно-логарифмический алгоритм может вычислять операцию O(log n) n раз. В нотации «О большое» линейно-логарифмический алгоритм выражается как O(n log n). Линейно-логарифмические алгоритмы часто разделяют набор данных на меньшие части и каждую из них обрабатывают по отдельности. Например, большинство наиболее эффективных алгоритмов сортировки, такие как сортировка слиянием, являются линейно-логарифмическими.

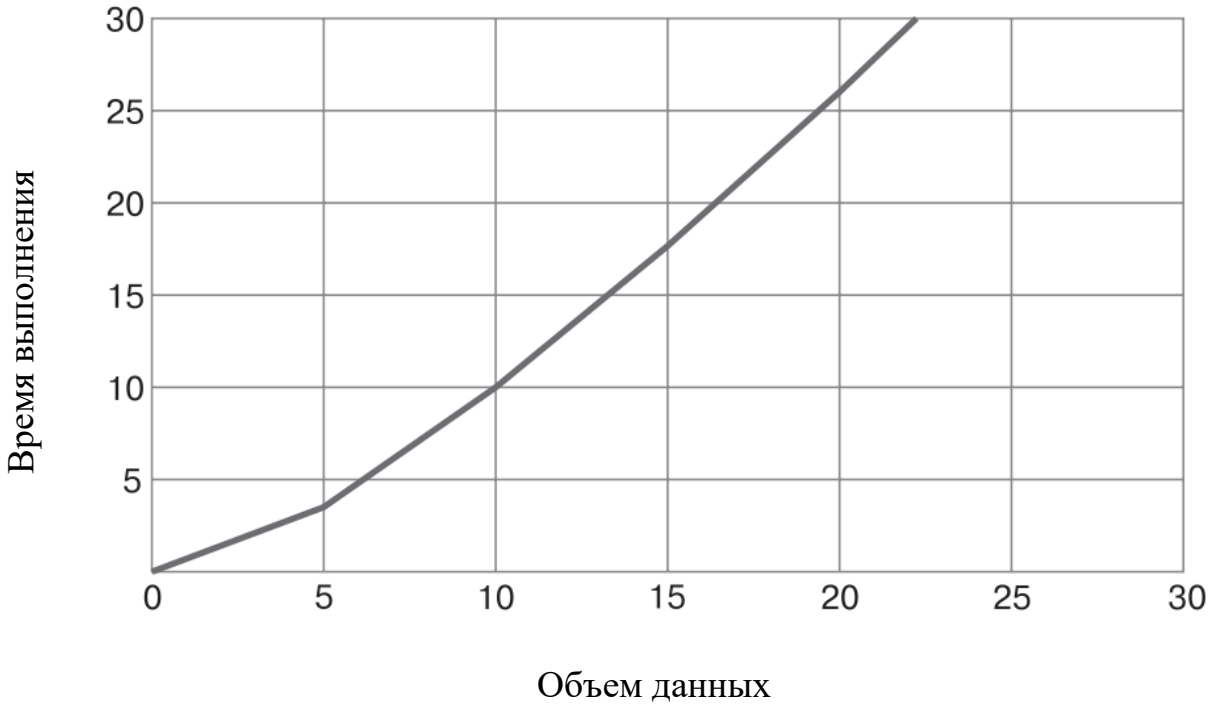


Рисунок 5 – Линейно-логарифмическая временная сложность [9]

Квадратичное время

Следующим по эффективности после линейно-логарифмического идет квадратичное время. Алгоритм выполняется за квадратичное время, когда его производительность прямо пропорциональна размеру задачи в квадрате. В нотации «О большое» квадратичный алгоритм выражается как O(n**2).

На графике алгоритма с квадратичной временной сложностью количество шагов резко увеличивается по мере увеличения размера задачи (рисунок 6).

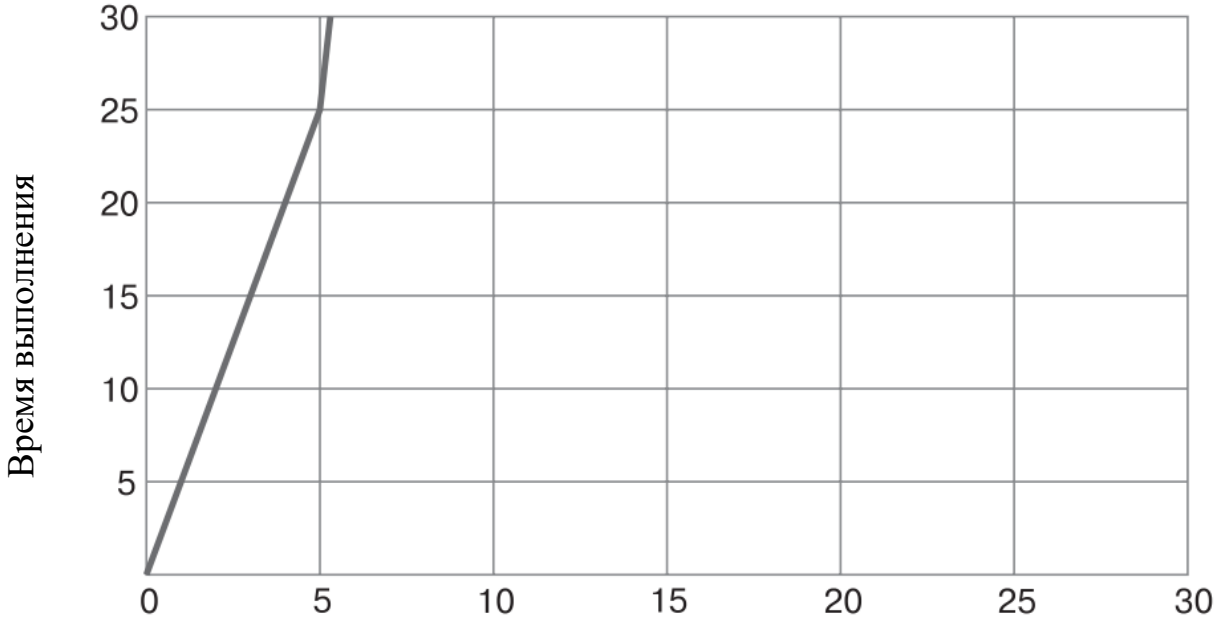


Рисунок 6 – Квадратичная временная сложность [9]

Как правило, если ваш алгоритм содержит два вложенных цикла, выполняемых от 1 до n (или от 0 до n – 1), временн я сложность будет как минимум O(n**2). Большинство алгоритмов сортировки, такие как сортировка методом вставок или пузырьковая сортировка, выполняются за квадратичное время.

Кубическое время

За квадратичной временной сложностью следует кубическая. Алгоритм выполняется за кубическое время, когда производительность прямо пропорциональна размеру задачи в кубе. В нотации «О большое» вы выражаете кубический алгоритм как O(n**3). Алгоритм с кубической сложностью похож на квадратичный, только n возводится в третью степень, а не во вторую.

Как и в алгоритме с квадратичной сложностью, наиболее важная часть данного уравнения — n**3, которая растет так быстро, что остальная часть уравнения, даже та, что включает в себя n**2, становится нерелевантной.

Таким образом, в нотации «О большое» кубическая сложность выглядит так: O(n) = n**3

Два вложенных цикла свидетельствуют о квадратичной временной сложности, а три вложенных цикла, выполняемых от 0 до n, — признак алгоритма с кубическим временем.

И квадратичная, и кубическая временные сложности представляют собой частные случаи полиномиальной временный сложности. Алгоритм, который выполняется за полиномиальное время, вычисляется как O(n**a), где a = 2 для квадратичного времени и a = 3 для кубического времени. При создании алгоритма, как правило, пытаются избежать полиномиального масштабирования, потому что по мере увеличения n резко возрастает время выполнения алгоритма. [9]

Экспоненциальное время

Алгоритм, который выполняется за экспоненциальное время, содержит константу, увеличенную до размеров задачи. Другими словами, алгоритму с экспоненциальным временем для завершения требуется с шагов, возведенных в степень n. Нотация «О большое» для экспоненциального времени равна O(c**n), где c — это константа. Значение константы не играет роли. Важно лишь то, что n находится в экспоненте.

Одним из примеров данной сложности, связанной с попыткой угадать числовой пароль из n-го количества десятичных знаков путем проверки всех возможных комбинаций, является O(10**n).

Отгадывание пароля — это пример алгоритма полного перебора, проверяющего все возможные варианты. Алгоритмы полного перебора, как правило, неэффективны, и выбирать их следует только в крайнем случае.

На рисунке 7 представлено сравнение эффективности рассмотренных видов алгоритмов.

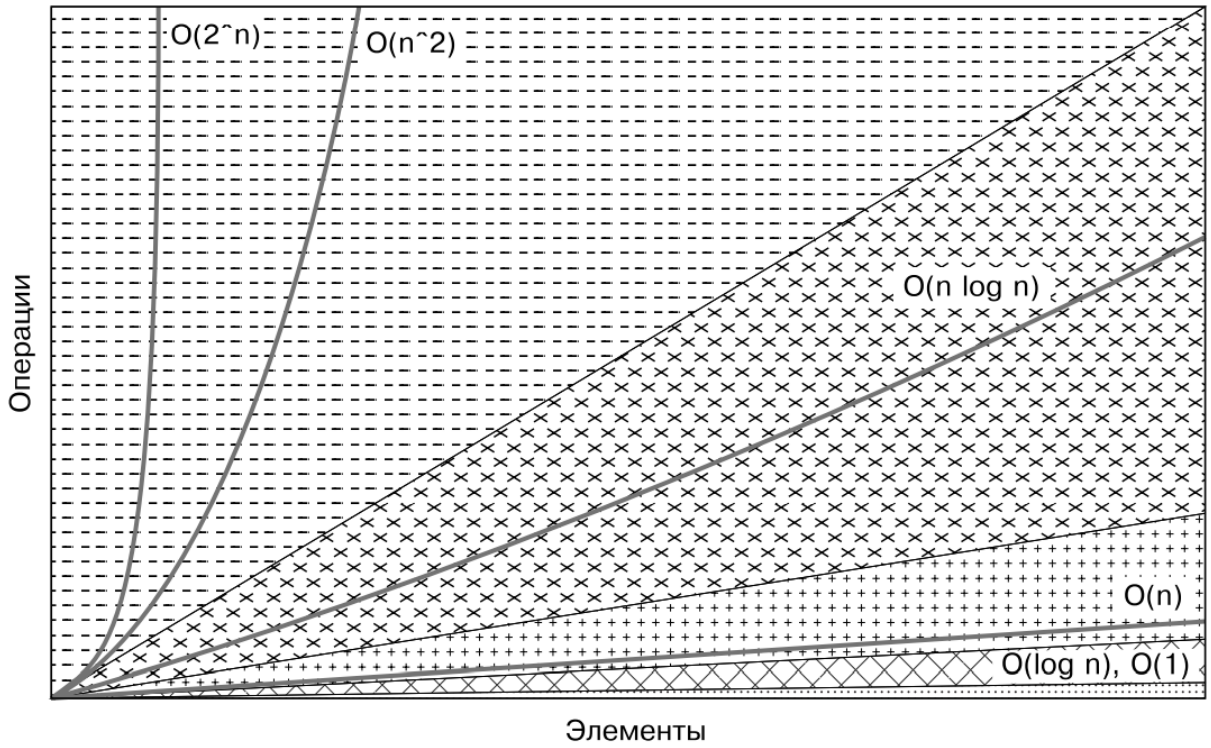


Рисунок 7 – Сравнение эффективности разных видов алгоритмов [9]

Из этого следует, что необязательно опираться на детали реализации алгоритма, такие как язык программирования или вычислительные способности оборудования, для оценки его временной эффективности. Благодаря О-нотации есть возможность распределить алгоритмы на категории в зависимости от их эффективности, независимо от прочих факторов (входные данные, детали реализации и т.д.)

Зная скорость роста алгоритма, можно прогнозировать, во сколько раз увеличится время выполнения алгоритма при кратном увеличении размерности задачи. [8, 10, 11]

1.5 Хранение и работа с данными

Многие программы каким-то образом сохраняют данные: они могут записывать их в файлы конфигурации, в отдельные файлы и хранить их где-то на устройстве пользователя или использовать хранилища – базы данных.

В этом разделе будет рассказано об использованной при выполнении проекта СУБД SQLite, а также инструменте для работы с базами данных ORM Entity Framework Core.

1.5.1 Система управления базами данных SQLite

SQLite – это лекговесная встраиваемая система управления базами данных. В данный момент она применяется в большей части приложений почти под каждую платформу. Её отличием от других СУБД является отказ от сервера БД – SQLite обращается сразу к представляющему базу данных. Соответственно, один из плюсов SQLite состоит в том, что ее не нужно настраивать перед работой.

SQLite представляет собой урезанную в возможностях SQL СУБД, однако обладает всеми основными возможностями. SQLite использует свой диалект языка запросов SQL, которая не на много отличается от прочих его реализаций, предлагаемых другими СУБД.

Благодаря отказу от необходимости иметь сервер базы данных для работы, для SQLite открылась возможность быстро осуществлять с одной платформы на другую без выполнения дополнительных операций – файл базы данных можно без потерь перенести на другое устройство, работающее под другой ОС.

SQLite необычайно удобный инструмент, позволяющий хранить данные портативно, что необходимо при работе многих приложений. Благодаря своей легкости он получил широкое распространение в сообществе разработчиков программного обеспечения, благодаря чему SQLite сейчас поддерживается многими языками программирования. [7]

1.5.2 ORM Entity Framework Core

Технология Entity Framework (EF) была впервые выпущена как часть .NET Framework 3.5 с Service Pack 1 в конце 2008 года. С тех пор эта технология значительно развилась, поскольку компания Microsoft пристально

наблюдала за тем, как программисты используют этот инструмент объектно-реляционного отображения (object-relational mapping, ORM) при работе над своими продуктами.

Суть ORM в определении сопоставления между столбцами в таблицах и свойствами в классах. Затем разработчик может взаимодействовать с объектами различных типов привычным для него способом вместо того, чтобы изучать способы сохранения значений в реляционной таблице или другой структуре, предоставляемой хранилищем данных NoSQL.

Entity Framework 6 (EF6) — это версия EF, включенная в последнюю версию .NET Framework. Она полная, стабильная и поддерживает способ определения модели EDMX (XML-файл), а также сложные модели наследования и некоторые другие расширенные функции.

Версия EF 6.3 и более поздние версии были извлечены из .NET Framework в виде отдельного пакета, поэтому данные версии могут поддерживаться в .NET Core 3.0 и более поздних версиях. Это позволяет переносить существующие проекты, такие как веб-приложения и сервисы, и запускать их на разных платформах. Тем не менее EF6 следует считать устаревшей технологией, поскольку она имеет некоторые ограничения при работе с разными платформами и в нее не будут добавляться новые функции.

По-настоящему кросс-платформенная версия EF Core отличается от устаревшей Entity Framework. Последняя версия EF Core — 6.0, соответствующая .NET 6.0.

EF Core 5 и более поздние версии поддерживают только .NET 5 и более поздние версии. EF Core 3.0 и более поздние версии работают только на платформах, поддерживающих .NET Standard 2.1, то есть .NET Core 3.0 и более поздние версии. Она не будет работать на платформах .NET Standard 2.0, таких как .NET Framework 4.8.

Помимо традиционных РСУБД, EF Core поддерживает современные облачные, нереляционные хранилища данных без схемы, такие как Microsoft Azure Cosmos DB и MongoDB, иногда со сторонними поставщиками.

Существует два подхода к работе с EF Core:

1. Сначала база данных — база данных уже существует, поэтому вы создаете модель, соответствующую ее структуре и функциональным возможностям.

2. Сначала код — базы данных не существует, поэтому вы создаете модель, а затем используете EF Core для создания базы данных, соответствующей ее структуре и функциональным возможностям.

EF Core позволяет разработчикам взаимодействовать с данными из различных баз данных как с объектами классов, построенных в соответствии с их представлением в БД, таким образом набор строк какой-то таблицы превращается в коллекцию объектов, которые программист может использовать в удобной ему форме, не проводя лишних преобразований. Объекты, построенные на основе данных из базы данных, называются сущностями и хранятся в специальных коллекциях, поддерживающих работу таких инструментов как LINQ.

Подобно .NET Core инфраструктура Entity Framework Core представляет собой полностью переписанную инфраструктуру Entity Framework 6. Она построена на основе .NET Core, давая возможность инфраструктуре EF Core функционировать на множестве платформ. Переписывание EF Core позволило добавить к EF Core новые средства и улучшения в плане производительности, которые не получилось бы разумно реализовать в EF 6.

Воссоздание целой инфраструктуры с нуля требует внимательного анализа того, какие средства будут поддерживаться новой инфраструктурой, а от каких придется отказаться. Одним из средств EF 6, которые отсутствуют в EF Core (и вряд ли когда-либо будут добавлены), является поддержка визуального конструктора сущностей (Entity Designer). В EF Core поддерживается парадигма разработки “сначала код”.

С каждым новым выпуском в инфраструктуру EF Core добавлялись дополнительные средства, которые присутствовали в EF 6, плюс совершенно новые средства, не существующие в EF 6. С выходом выпуска 3.1 список

важных функций, отсутствующих в EF Core (в сравнении с EF 6), был значительно уменьшен, а с выходом выпуска 5.0 разрыв сократился еще больше. Фактически инфраструктура EF Core располагает всем необходимым для большинства проектов.

“За кулисами” EF Core использует инфраструктуру ADO.NET. Подобно любому взаимодействию ADO.NET с хранилищем данных EF Core применяет для этого поставщик данных ADO.NET. Прежде чем поставщик данных ADO.NET можно будет использовать в EF Core, его потребуется обновить для полной интеграции с EF Core. Из-за такой добавленной функциональности доступных поставщиков данных EF Core может оказаться меньше, чем поставщиков данных ADO.NET.

Преимущество инфраструктуры EF Core, применяющей шаблон поставщиков баз данных ADO.NET, заключается в том, что она позволяет объединять в одном проекте парадигмы доступа к данным EF Core и ADO.NET, расширяя ваши возможности. Например, в случае использования EF Core с целью предоставления подключения, схемы и имени таблицы для операций массового копирования задействуются возможности сопоставления EF Core и функциональность программы массового копирования, встроенная в ADO.NET.

Инфраструктура EF Core лучше всего вписывается в процесс разработки в случае применения подходов в стиле “формы поверх данных” (или “API-интерфейс поверх данных”). Оптимальными для EF Core являются операции над небольшим количеством сущностей, использующие шаблон единицы работы с целью обеспечения согласованности. Она не очень хорошо подходит для выполнения крупномасштабных операций над данными вроде тех, что встречаются приложениях хранилищ данных типа “извлечение, трансформация, загрузка” (extract-transform-load — ETL) или в больших системах построения отчетов.

К главным компонентам EF Core относятся DbContext, ChangeTracker, специализированный тип коллекции DbSet, поставщики баз данных и сущности приложения.

Класс DbContext входит в состав главных компонентов EF Core и предоставляет доступ к базе данных через свойство Database. Объект DbContext управляет экземпляром ChangeTracker, поддерживает виртуальный метод OnModelCreating() для доступа к текучему API-интерфейсу (Fluent API), хранит все свойства DbSet<T> и предлагает метод SaveChanges(), позволяющий сохранять данные в хранилище. Он применяется не напрямую, а через специальный класс, унаследованный от DbContext. Именно в этом классе размещены все свойства типа DbSet<T>.

Экземпляр DbContext конфигурируется с использованием экземпляра класса DbContextOptions. Экземпляр DbContextOptions создается с применением DbContextOptionsBuilder, т.к. класс DbContextOptions не рассчитан на создание экземпляров непосредственно в коде. Через экземпляр DbContextOptionsBuilder выбирается поставщик базы данных (наряду с любыми настройками, касающимися поставщика) и устанавливаются общие параметры экземпляра DbContext инфраструктуры EF Core (наподобие ведения журнала). Затем свойство Options внедряется в базовый класс DbContext во время выполнения.

Такая возможность динамического конфигурирования позволяет изменять настройки во время выполнения, просто выбирая разные параметры (скажем, поставщик MySQL вместо SQL Server) и создавая новый экземпляр производного класса DbContext.

Для каждой сущности в своей объектной модели вы добавляете свойство типа DbSetcTX Класс DbSet<T> представляет собой специализированную коллекцию, используемую для взаимодействия с поставщиком баз данных с целью получения, добавления, обновления и удаления записей в базе данных. Каждая коллекция DbSet<T> предлагает несколько основных служб для взаимодействия с базой данных. Любые запросы LINQ, запускаемые в

отношении класса DbSet<T>, транслируются поставщиком базы данных в запросы к базе данных.

Класс DbSet<T> реализует интерфейс IQueryable<T> и обычно является целью запросов LINQ to Entity. Помимо расширяющих методов, добавленных инфраструктурой EF Core, класс DbSet<T> поддерживает расширяющие методы, такие как ForEach(), Select() и All().

Типы запросов — это коллекции DbSet<T>, которые применяются для изображения представлений, оператора SQL или таблиц без первичного ключа. В предшествующих версиях EF Core для всего упомянутого использовался тип DbQuery<T>, но начиная с EF Core 3.1, тип DbQuery больше не употребляется. Тйпы запросов добавляются к производному классу DbContext с применением свойств DbSet<T> и конфигурируются как не имеющие ключей. Остальные действия по конфигурированию производятся в Fluent API.

Экземпляр ChangeTracker отслеживает состояние объектов, загруженных в DbSet<T> внутри экземпляра DbContext.

Экземпляр ChangeTracker способен генерировать два события: StateChanged и Tracked. Событие StateChanged инициируется в случае изменения состояния сущности. Оно не генерируется при отслеживании сущности в первый раз. Событие Tracked инициируется, когда сущность начинает отслеживаться, либо за счет добавления экземпляра DbSet<T> в коде, либо при возвращении из запроса.

2 РАЗРАБОТКА ПРОТОТИПА ПРОГРАММЫ

Данная глава будет посвящена процессу разработки прототипа программы. В ней будет описано устройство разработки, принцип её работы.

2.1 Функционал приложения

Перед началом разработки необходимо определить функционал прототипа. Итак, программа должна иметь возможность:

– подключаться к осциллографу и обмениваться с ним данными (отправлять ему команды и получать ответ);

– запрашивать необходимые данные;

– обрабатывать и хранить получаемые данные;

– визуализировать выбранный набор данных;

– подключить сторонние скрипты для обработки определенного набора данных.

В соответствии с необходимым функционалом можно разделить прототип на 5 частей:

1. Ядро приложения, отвечающее за логику обмена информацией с осциллографом.
2. Модуль, содержащий в себе логику запросов к осциллографу.
3. Модуль, отвечающий за взаимодействие с базой данных приложения.
4. Приложение с графическим интерфейсом, также содержащее в себе модуль визуализации данных.
5. Модуль взаимодействия со сторонними скриптами.

2.1.1 Ядро приложения

Ядро приложения не должно иметь большого объема, все что нужно – функция подключения к нужному устройству и открытия потока между устройствами, методы отправки команды и чтения ответа. Из того следует, что данный модуль будет содержать лишь один класс – LxiClient.

На рисунке 8 представлено краткое описание класса LxiClient и всех его методов.

```
/*
 * Класс, содержащий функции для взаимодействия с осциллографом
 * Связь устававливается с помощью классов TcpClient и NetworkStream
 */
public class LxiClient
{
    public LxiClient() { }

    /*
     * Метод отправки команды
     * Преобразует строковую команду в массив байтов и отправляет его
     */
    public void SendCommand(string command, NetworkStream ns)...

    /*
     * Метод чтения ответа
     * Считывает массив байтов и преобразует в строковое значение
     * Применяется для получения преамбулы
     */
    public string GetResponse(NetworkStream ns)...

    /*
     * Метод чтения ответа
     * Считывает массив байтов заданного размера, не преобразовывая
     * Применяется для чтения семплов
     */
    public byte[] GetByteResponse(int responseSize, NetworkStream ns)...

    /*
     * Перегрузка метода чтения ответа
     * Считывает массив байтов заданного размера, не преобразовывая
     * Применяется для чтения семплов
     * Принимает размер ответа и размер буфера для приема данных
     */
    public byte[] GetByteResponse(int responseSize, int bufferSize, NetworkStream ns)...
```

Рисунок 8 – Описание класса LxiClient

Как видно из рисунка данный модуль имеет лишь три уникальных метода и одну перегрузку. Однако этот класс является основой всего приложения, поскольку в нем сосредоточена логика взаимодействия с осциллографом.

2.1.2 Модуль взаимодействия с базой данных

Данный модуль содержит логику взаимодействия с СУБД SQLite посредством ORM Entity Framework Core.

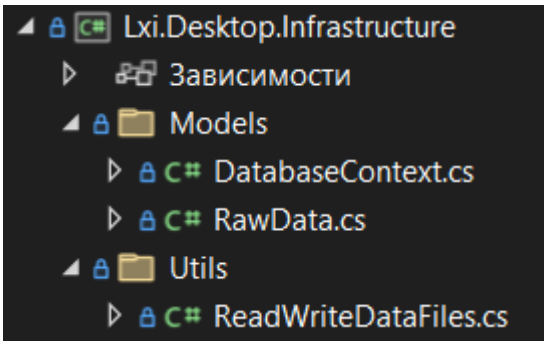


Рисунок 9 – Модуль Infrastructure

Разработка модели данных

Перед тем как начинать пытаться получить данные было необходимо определить класс, описывающий получаемые данные – создать «контейнер», в который эти данные будут складываться, чтобы в последствии данные, полученные в результате удачных попыток, не улетали в пустоту.

Для того, чтобы правильно принять и обработать информацию, в первую очередь необходимо получить и сохранить преамбулу, содержащую информацию об отображаемых на экране осциллографа данных (рисунок 10).

Return Format

The query returns 10 waveform parameters separated by ",":

<format>,<type>,<points>,<count>,<xincrement>,<xorigin>,<xreference>,<yincrement>,<yorigin>,<yreference>

Wherein,
<format>: 0 (BYTE), 1 (WORD) or 2 (ASC).
<type>: 0 (NORMal), 1 (MAXimum) or 2 (RAW).
<points>: an integer between 1 and 12000000. After the memory depth option is installed, <points> is an integer between 1 and 24000000.
<count>: the number of averages in the average sample mode and 1 in other modes.
<xincrement>: the time difference between two neighboring points in the X direction.
<xorigin>: the start time of the waveform data in the X direction.
<xreference>: the reference time of the data point in the X direction.
<yincrement>: the waveform increment in the Y direction.
<yorigin>: the vertical offset relative to the "Vertical Reference Position" in the Y direction.
<yreference>: the vertical reference position in the Y direction.

Рисунок 10 – Данные передаваемые в преамбуле

Преамбула несет в себе информацию о формате получаемых данных (в работе использовался формат BYTE), типе получаемых данных (в работе используется тип RAW), количестве передаваемых точек, начальной точке по осям X и Y, смещении по осям X и Y, приращении значений по осям X и Y.

При получении массива данных стоит учитывать, что первые 11 байт информации являются данными заголовка (рисунок 11), поэтому при обработке они будут отметаться.

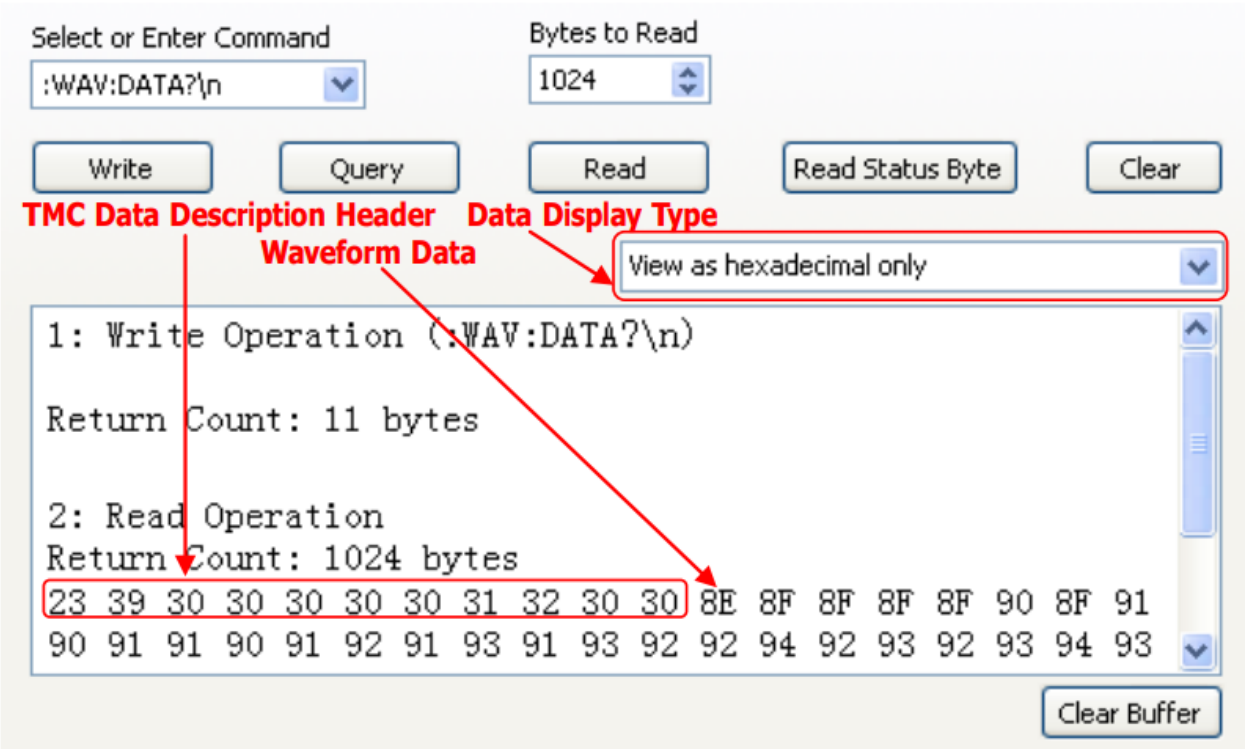
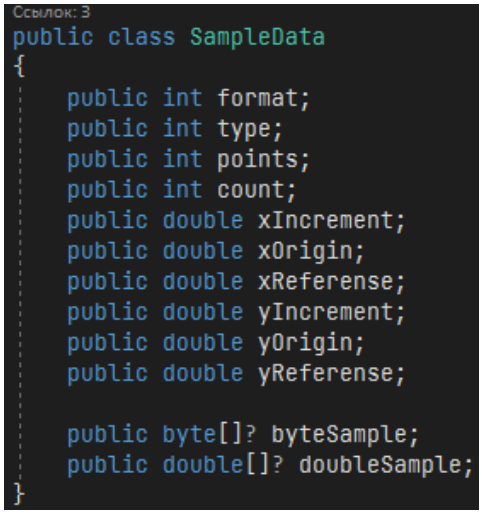


Рисунок 11 – Пример получаемых данных

Из этого следовало, что базово модель данных должна включать в себя 11 полей: 10 полей из преамбулы и одно поле, являющееся массивом байтов информации о сэмплах (рисунок 14). Первоначально было принято решение добавить к определению класса также поле для хранения данных, преобразованных в вещественные числа двойной точности, но в связи с тем, что СУБД имеет плохую производительность при работе с большими

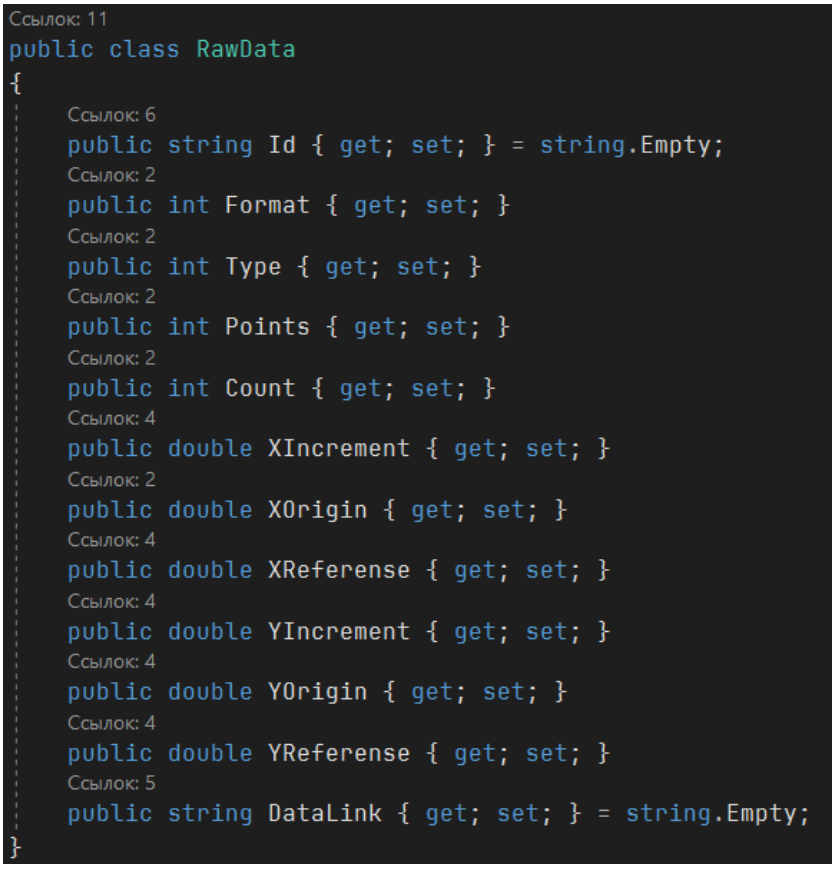
объемами данных, модель была преобразована в представленную на рисунке 12.



```
Ссылок: 3
public class SampleData
{
    public int format;
    public int type;
    public int points;
    public int count;
    public double xIncrement;
    public double xOrigin;
    public double xReferense;
    public double yIncrement;
    public double yOrigin;
    public double yReferense;

    public byte[]? byteSample;
    public double[]? doubleSample;
}
```

Рисунок 12 – Модель данных



```
Ссылок: 11
public class RawData
{
    Ссылок: 6
    public string Id { get; set; } = string.Empty;
    Ссылок: 2
    public int Format { get; set; }
    Ссылок: 2
    public int Type { get; set; }
    Ссылок: 2
    public int Points { get; set; }
    Ссылок: 2
    public int Count { get; set; }
    Ссылок: 4
    public double XIncrement { get; set; }
    Ссылок: 2
    public double XOrigin { get; set; }
    Ссылок: 4
    public double XReferense { get; set; }
    Ссылок: 4
    public double YIncrement { get; set; }
    Ссылок: 4
    public double YOrigin { get; set; }
    Ссылок: 4
    public double YReferense { get; set; }
    Ссылок: 5
    public string DataLink { get; set; } = string.Empty;
}
```

Рисунок 13 – Переработанная модель данных

Теперь сведения о наборе данных, получаемые в преамбуле, хранятся в таблице базы данных, а показания осциллографа в необработанном виде (в

виде набора байтов) сохраняются в отдельном файле, ссылка на который хранится в поле DataLink представленной модели.

Для чтения и записи байтовых данных в файл и был разработан класс ReadWriteDataFiles. Как понятно из названия этот класс содержит набор методов для чтения и записи данных в файлы, его определение представлено на рисунке 14.

```
Ссылок: 5
public static class ReadWriteDataFiles
{
    // Метод для создания папки, в которой будут храниться файлы с данными
    Ссылок: 1
    public static string CreateSubDirectory(string target, string subDirName)...
    // Метод для создания файла, в котором будут храниться данные
    Ссылок: 1
    public static string CreateDataFile(string target, int channel)...
    // Метод для записи данных в указанный файл
    Ссылок: 1
    public static bool WriteData(string path, in byte[] bytes)...
    // Метод для чтения данных из указанного файла
    Ссылок: 2
    public static bool ReadData(string path, out byte[] bytes)...
    // Вспомогательный метод для проверки доступа к файлу для предотвращения исключений
    Ссылок: 2
    public static bool CheckAccess(FileInfo file, FileAccess access)...
}
```

Рисунок 14 – Определение класса ReadWriteDataFiles

2.1.3 Модуль запросов

Данный модуль уже не ограничится одним классом, поскольку будет содержать как код команды осциллографу, так и некоторые другие функции (рисунок 15).

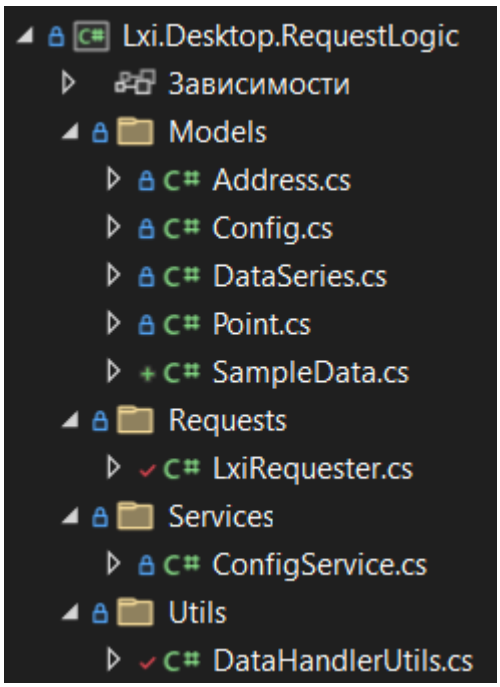


Рисунок 15 – Модуль RequestLogic

В классе LxiRequester определена логика запросов к осциллографу (рисунок 16). Он содержит конструктор (LxiRequester), метод, устанавливающий соединение с нужным прибором (SetConnection), а также метод закрывающий ранее установленное соединение (CloseConnection), метод получения данных с одного из каналов вывода данных прибора (GetSample), который используется в методе последовательного чтения с нескольких каналов (ReadFromChannels), а также утилитарный метод, предназначенный для обработки преамбулы к приходящим данным (PreambleParse).

В классе DataHandlerUtils содержится логика преобразования данных из байтового представления в вещественное, двойной точности, а также алгоритм прореживания данных.

Класс ConfigService используется для чтения данных из файла конфигурации (рисунок 17). Используя прочитанные данные, он генерирует класс Config, который затем используется другими классами.

```
public class LxiRequester
{
    private LxiClient lxiClient;
    private Config cfg;
    private DatabaseContext db;
    private TcpClient tcpClient;
    private int maxBytesPerRequest = 250000;
    private int bufferSize = 8192;

    Ссылок: 0
    public LxiRequester(LxiClient lxiClient, ConfigService cfg,
        DatabaseContext ctx)
    {
        this.lxiClient = lxiClient;
        this.cfg = cfg.config;
        this.db = ctx;
    }
    Ссылок: 1
    public void SetConnection(string host, int port)...
    Ссылок: 1
    public void CloseConnection()...
    Ссылок: 1
    public async Task<List<string>> ReadFromChannels(int[] ch, Address? address = null)...
    Ссылок: 1
    private async Task<string> CreateRawDataRecord(SampleData data, int ch)...
    /*
     * Метод получения полной модели данных с указанного канала
     * Считывает преамбулу и семплы, формируя из этих данных
     * единый объект
     */
    Ссылок: 1
    private SampleData GetSample(int channel, NetworkStream ns)...
    /*
     * Метод парсинга преамбулы
     */
    Ссылок: 1
    private void PreambleParse(SampleData sampleData, string preamble)...
```

Рисунок 16 – Определение класса LxiRequester

```
{
  "TargetDirectory": "C:\\Users\\Vladimir\\Desktop\\Samples",
  "Addresses": [
    {
      "ip": "192.168.88.50",
      "port": 5555
    },
    {
      "ip": "192.168.88.51",
      "port": 5555
    },
    {
      "ip": "192.168.88.52",
      "port": 5555
    },
    {
      "ip": "192.168.88.53",
      "port": 5555
    },
```

Рисунок 17 – Структура файла конфигурации

2.1.4 Пользовательский интерфейс

Пользовательский интерфейс определен с использованием технологии Windosw Forms и содержит классы представленные на рисунке 18.

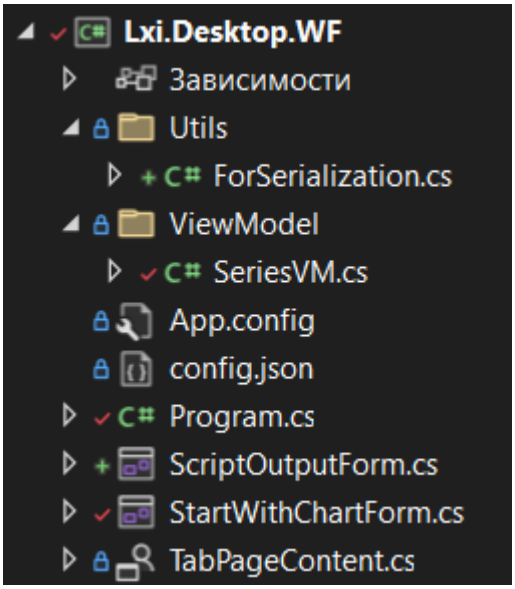


Рисунок 18 – Модуль графического интерфейса

Метод Main класса Program является точкой входа в приложения (Рисунок 19), в нем производится вызов методов конфигурирования приложения перед запуском, а также запуск стартовой формы.

```
Ссылок: 0
internal static class Program
{
    private static ServiceCollection services = new ServiceCollection();
    public static ServiceProvider? provider;


    /// <summary>
    ///  The main entry point for the application.
    /// </summary>
    [STAThread]
    Ссылок: 0
    static void Main()
    {
        // To customize application configuration such as set high DPI settings or default font,
        // see https://aka.ms/applicationconfiguration.
        ApplicationConfiguration.Initialize();

        provider = ConfigureServices(services);

        Application.Run(provider.GetRequiredService<StartWithChartForm>());
    }
```

Рисунок 19 – Метод Main класса Program

Объекты классов ServiceCollection и ServiceProvider – services и provider соответственно, являются частью реализации подхода внедрение зависимостей. В методе ConfigureServices (рисунок 20), также реализованном в классе Program, происходит добавление необходимых классов в контейнер зависимостей для дальнейшего использования. К этим классам относятся StartWithChartForm, являющаяся основным рабочим окном, LxiClient и LxiRequester, необходимые для взаимодействия с осциллографом, а также DatabaseContext, класс необходимый для взаимодействия с базой данных.

```
Ссылок: 1
static ServiceProvider ConfigureServices(IServiceCollection s)
{
    s.AddSingleton<ConfigService>()
        .AddTransient<StartWithChartForm>()
        .AddTransient<LxiClient>()
        .AddTransient<LxiRequester>()
        .AddDbContext<DatabaseContext>(options ⇒
            options.UseSqlite(ConfigurationManager
                .ConnectionStrings["Sqlite"].ConnectionString));

    return s.BuildServiceProvider();
}
```

Рисунок 20 – Метод ConfigureServices

Основное рабочее окно имеет вид, представленный на рисунке 21. Условно его можно разделить на две части – слева область для взаимодействия с осциллографом (в разделе “Get samples” пользователь может выбрать нужный осциллограф по ip, выбрать из каких каналов будет происходить чтение данных), уже считанными данными (в разделе “Show samples” пользователь может выбрать какие наборы данных визуализировать), а также выбрать скрипт для отработки (раздел “Choose script”); часть справа предназначена для визуализации выбранных наборов данных

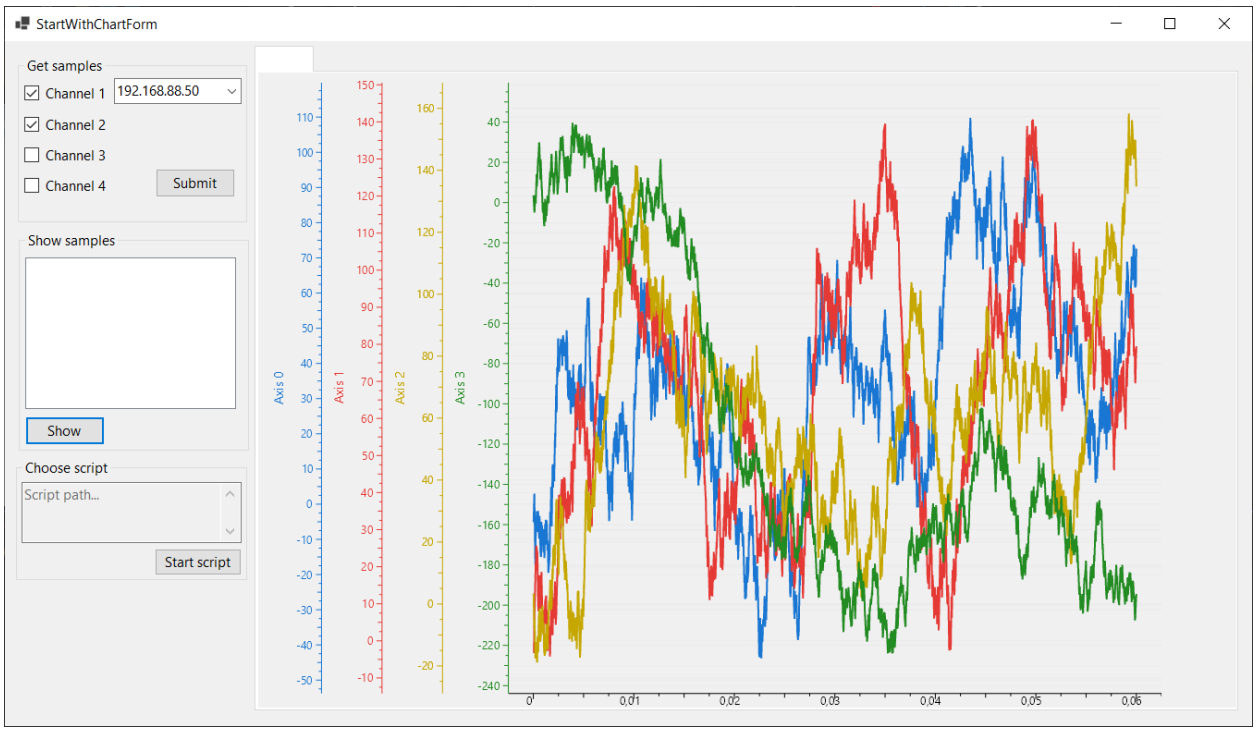


Рисунок 21 – Основное рабочее окно

Правая часть рабочего окна содержит контроллер вкладок – при выборе нового набора данных для визуализации открывается новая вкладка с построенным на ней графиком. Возможность быстро переключаться от одного набора данных к другим делает программу намного удобнее, а также избавляет от необходимости совершать лишние расчеты.

Каждая новая вкладка содержит в себе экземпляр компонента “TabPageContent”, который имеет ViewModel, определяющий логику построения и обновления графиков и представленный классом SeriesVM (рисунок 22). Большая часть логики содержится в конструкторе класса – в нем происходит инициализация объектов графиков, которые позже размещаются в окне приложения. Метод ScaleLines отвечает за масштабирование графика, а в методе PrepareDataForScriptWork происходит подготовка имеющихся на экране данных к передаче их стороннему скрипту для обработки.

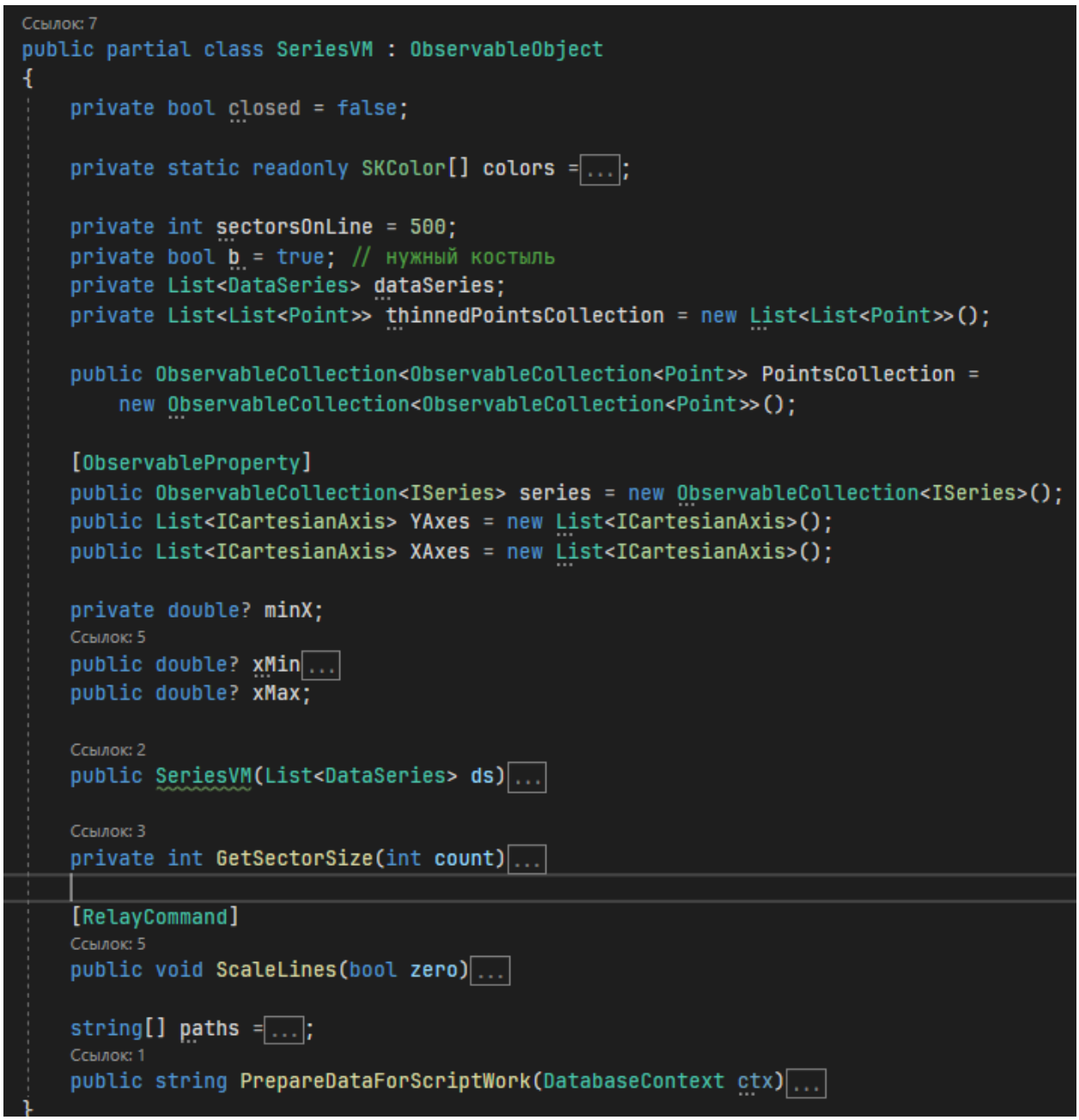


Рисунок 22 – Класс SeriesVM

2.1.5 Модуль взаимодействия со сторонними скриптами

В качестве языка программирования, скрипты которого будут использоваться, был выбран Python, потому что он популярен, прост в использовании и имеет множество математических библиотек. Python часто применяют для анализа больших данных, поэтому в данном случае его использование оправдано.

Данный модуль представлен лишь одним классом – PyRunner (рисунок 23).

Рисунок 23 – Класс PyRunner

Этот класс содержит три метода. Для запуска работы скрипта применяется метод Execute, который сначала запускает метод CreateArgsString для получения строки аргументов необходимых скрипту в работе, потом запускает метод RunScript, который и инициализирует процесс, передавая ему строку аргументов.

По завершении работы скрипта программа должны получить строку содержащую ссылку на файл с результатами работы, который позднее может быть выведен пользователю программы.

Посмотреть код программы можно по ссылке: https://bitbucket.org/VelderGoose/lxidesktop/src/master/

2.2 Тестирование разработанной программы

Тестирование разрабатываемого продукта необходимо для того, чтобы быть уверенным в его работоспособности, а также для своевременного исправления имеющихся ошибок.

Для проведения тестирования программы необходимо:

1. Сгенерировать объем данных произвольного размера (в подходящем формате) и отправить в программу.
2. Снять показания с реального оборудования.

В первую очередь приступим к тестированию с помощью генерации данных.

2.2.1 Тестирование сгенерированными данными

Для этих целей разработан метод, который создает несколько массивов вещественных чисел двойной точности указанного размера (10 млн. значений), которые затем поступают в функцию визуализации.

Результат работы функции для тестирования представлен на рисунке 24.

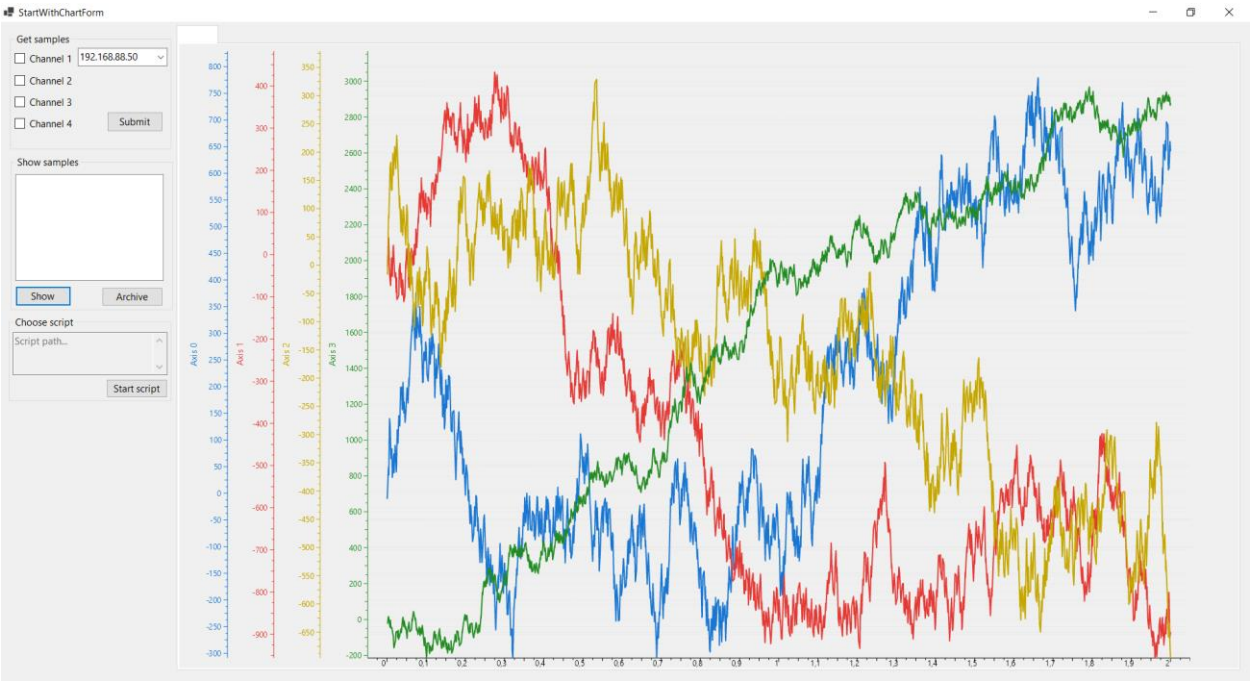


Рисунок 24 – Результаты работы метода для тестирования

2.2.2 Снятие показаний реального сварочного процесса

Снятие показаний проводилась в следующих условиях:

– вид сварки: ручная дуговая покрытым электродом;

– основной металл: пластина из стали 09Г2С толщиной 10 мм;

– электрод: LB-52U диаметром 3,2 мм;

– сила тока: 90 А;

– время сварки: 90 сек.;

Замеры проводились по двум каналам – первый снимал силу тока, а второй напряжение (рисунки 25-27).

В ходе проведения показаний был получен валик наплавленного металла длиной 100 мм (рисунок 28).

Теперь, имея показания осциллографа, размеры наплавленного валика и другие данные о процессе, можно, используя модуль работы со сторонними скриптами, выполнить более подробный анализ имеющихся данных, например, узнать погонную энергию процесса, тепловую мощность дуги, абсолютное значение напряжения во время сварки и многое другое.

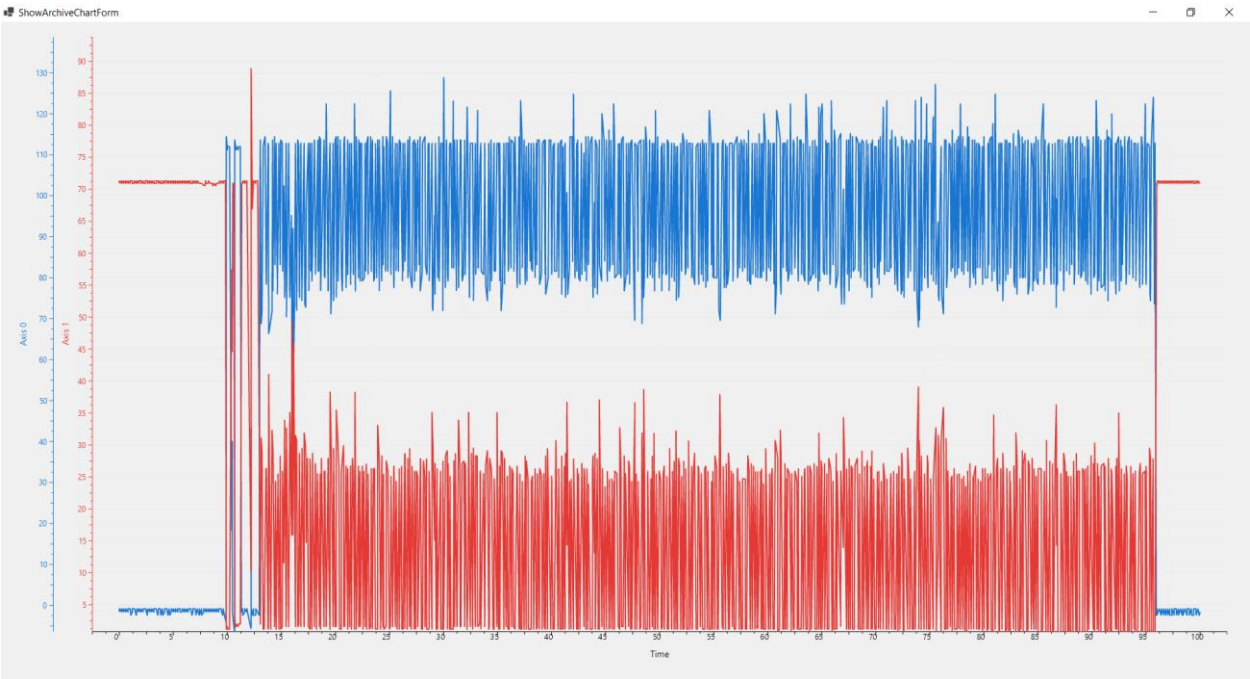


Рисунок 25 – Результат снятия показаний реального сварочного процесса: синий график – сила тока, красный – напряжение

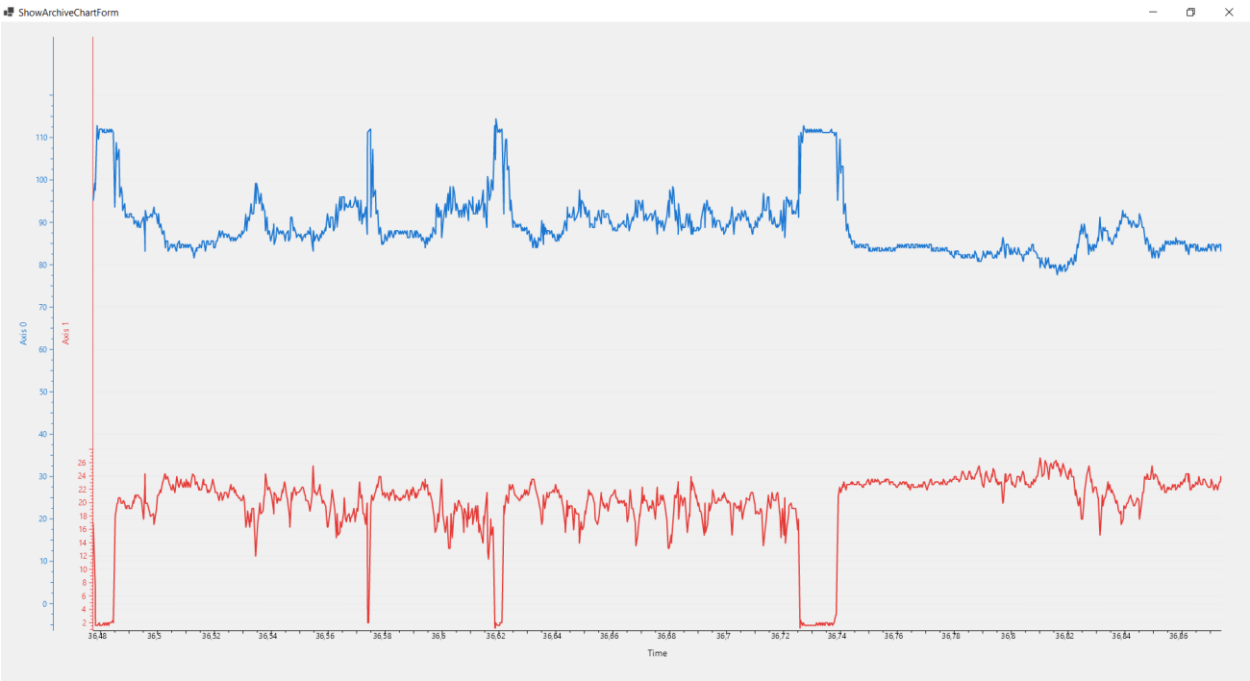


Рисунок 26 – Результат снятия показаний реального сварочного процесса, в приближении

Рисунок 27 – Результат снятия показаний реального сварочного процесса на осциллографе

Рисунок 28 – Полученный валик наплавленного металла

На рисунке 29 представлен простой скрипт для расчета среднего значения по графику, а на рисунке 30 результат его работы.

```python
import sys
import os
import json
print("Script start")
with open(sys.argv[1]) as jsonFile:
    jsonString = jsonFile.read()

preambles = json.loads(jsonString)
preLength = len(preambles)
for i in range(preLength):
    pre = preambles[i]
    length = os.path.getsize(pre['DataLink'])
    f = open(pre['DataLink'],'rb')
    bytes = []
    for i in range (length):
        y = int(f.read(1).hex(), 16)
        yOr = float(pre['YOrigin'])
        yRe = float(pre['YReferense'])
        yIn = float(pre['YIncrement'])
        handledData = (y - yOr - yRe) * yIn;
        bytes.append(handledData)
    f.close()
    average = sum(bytes)/len(bytes)
    id = pre['Id']
    print("Sample {id} has an average value", end=" ")
    print(average, end="\n")

print("Script end")
```

Рисунок 29 – Внешний скрипт для расчета среднего значения по графику

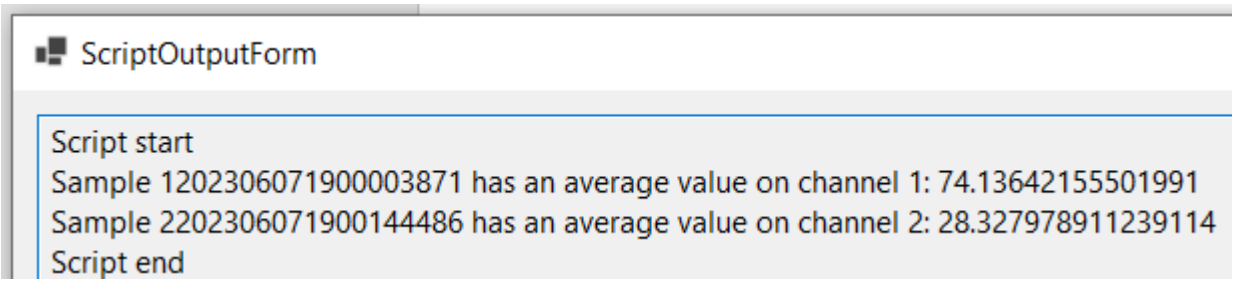


Рисунок 30 – Результат работы внешнего скрипта

ЗАДАНИЕ ДЛЯ РАЗДЕЛА
«ФИНАНСОВЫЙ МЕНЕДЖМЕНТ, РЕСУРСОЭФФЕКТИВНОСТЬ И РЕСУРСОСБЕРЕЖЕНИЕ»

Обучающемуся:

Группа	ФИО
1ВМ11	Иванову Владимиру Николаевичу

Школа	ИШНКБ	Отделение (НОЦ)	ОЭИ
Уровень образования	Магистратура	Направление/специальность	15.04.01 Машиностроение

Исходные данные к разделу «Финансовый менеджмент, ресурсоэффективность и ресурсосбережение»:	
1. *Стоимость ресурсов научно исследования (НТИ): материально-технических, энергетических, финансовых, информационных и человеческих* 2. *Нормы и нормативы расходования ресурсов* 3. *Используемая система налогообложения, ставки налогов, отчислений, дисконтирования и кредитования*	*Работа с информацией, представленной в российских и иностранных научных публикациях, аналитических материалах, статических бюллетенях и изданиях, нормативно-правовых документах;*
Перечень вопросов, подлежащих исследованию, проектированию и разработке:	
1. *Оценка коммерческого потенциала, перспективности и альтернатив проведения НИ с позиции ресурсоэффективности и ресурсосбережения*	*Проведение предпроектного анализа. Выполнение SWOT-анализа проекта*
2. *Разработка устава научно-технического проекта*	*Определение целей и ожиданий, требований проекта. Определение заинтересованных сторон и их ожиданий*
3. *Планирование процесса управления НИ: структура и график проведения, бюджет, риски и организация закупок*	*Составление календарного плана проекта. Определение бюджета НИ*
4. *Определение ресурсной (ресурсосберегающей), финансовой, бюджетной, социальной и экономической эффективности исследования*	*Проведение оценки экономической эффективности разработки программного обеспечения для анализа сварочного процесса*
Перечень графического материала *(с точным указанием обязательных чертежей)***:**	
1. *Матрица SWOT* 2. *Альтернативы проведения НИ* 3. *График проведения и бюджет НИ* 4. *Оценка ресурсной, финансовой и экономической эффективности НИ*	

Дата выдачи задания для раздела по линейному графику	

Задание выдал консультант:

Должность	ФИО	Ученая степень, звание	Подпись	Дата
профессор ОСГН	Гасанов Магеррам Али оглы	д.э.н., доцент		

Задание принял к исполнению обучающийся:

Группа	ФИО	Подпись	Дата
1В71	Иванов Владимир Николаевич		

3 ФИНАНСОВЫЙ МЕНЕДЖМЕНТ, РЕСУРСОЭФФЕКТИВНОСТЬ И РЕСУРСОСБЕРЕЖЕНИЕ

Данная научно-исследовательская работа направлена на разработку программного обеспечения для анализа различных параметров сварочного процесса, применимого в производстве и контроле качества выполняемых работ. Работы над проектом можно разделить на несколько видов: исследование возможностей применяемого инструментария, разработка программы или её модуля, тестирование с использованием сгенерированного набора случайных данных, тестирование с использованием данных реального сварочного процесса. Основная часть работы проводилась в учебном помещении НИ ТПУ.

Разработка решения производится группой квалифицированных работников, состоящей из двух человек – руководителя и студента-дипломника.

Целью раздела «Финансовый менеджмент, ресурсоэффективность и ресурсосбережение» является определение перспективности и успешности НТИ, оценка его эффективности, уровня возможных рисков, разработка механизма управления и сопровождения конкретных проектных решений на этапе реализации.

Для достижения обозначенной цели необходимо решить следующие задачи:

– Оценить коммерческий потенциал и перспективность разработки НТИ

– Осуществить планирование этапов выполнения исследования.

– Рассчитать бюджет затрат на исследования.

– Произвести оценку научно-технического уровня исследования и оценку рисков.

К научно-исследовательским работам относятся работы поискового, теоретического и экспериментального характера, которые выполняются с

целью расширения, углубления и систематизации знаний по определенной научной проблеме и создания научного задела.

3.1 Предпроектный анализ

3.1.1 Потенциальные потребители результатов исследования

Данная разработка может представлять интерес для организаций, работающих в области исследования сварочных процессов, разработки сварочного оборудования. Как правило компании разрабатывающие оборудование для сварки пытаются различными способами добиться улучшения сварочного процесса, предлагая своим потребителям новые методики сварки. Для своих целей они могут использовать разработанное решение. Поэтому можно говорить о том, что проект имеет высокий коммерческий потенциал.

3.1.2 Анализ конкурентных технических решений

Детальный анализ конкурирующих разработок, существующих на рынке, необходимо проводить систематически, поскольку рынки пребывают в постоянном движении. Такой анализ помогает вносить коррективы в научное исследование.

Анализ конкурентных технических решений с позиции ресурсоэффективности и ресурсосбережения позволяет провести оценку сравнительной эффективности научной разработки и определить направления для ее будущего повышения.

В ходе исследования были рассмотрены различные системы для сбора и анализа данных сварочного процесса и в данный момент у разработки имеется 1 явный конкурент. Конкурентный анализ будет проводится с помощью оценочной карты (таблица 1). Сравниваемые системы будут представлены следующим образом:

А – разработка, представленная в данной работе;

Б – Kemppi Arc System 2.5 – ArcInfo;

Критерии для сравнения и оценки ресурсоэффективности и ресурсосбережения, приведенные в таблице 1, подбираются, исходя из выбранных объектов сравнения с учетом их технических и экономических особенностей разработки и эксплуатации.

Позиция разработки и конкурентов оценивается по каждому показателю экспертным путем по пятибалльной шкале, где 1 – наиболее слабая позиция, а 5 – наиболее сильная. Веса показателей, определяемые экспертным путем, в сумме должны составлять 1.

Анализ конкурентных технических решений производится по формуле (1):

$$K = \sum B_i \cdot \text{Б}_i\,, \qquad (1)$$

где K – конкурентоспособность научной разработки или конкурента;

B_i – вес i-го показателя (в долях единицы);

Б_i – балл i-го показателя.

Таблица 1 – Оценочная карта для сравнения конкурентных технических решений (разработок)

Критерии оценки	Вес критерия	Баллы		Конкурентоспособность	
		$\text{Б}_\text{А}$	$\text{Б}_\text{Б}$	$\text{К}_\text{А}$	$\text{К}_\text{Б}$
1	2	3	4	5	6
Технические критерии оценки ресурсоэффективности					
1 Простота использования	0,1	2	4	0,2	0,4
2 Подробность предоставляемых данных	0,15	4	4	0,6	0,6
3 Уровень нагрузки на оборудование	0,15	3	5	0,45	0,75
4 Разнообразие доступных возможностей анализа	0,3	5	3	1,5	0,9
Экономические критерии оценки ресурсоэффективности					
1 Цена	0,15	5	3	0,75	0,45
2 Срок окупаемости	0,15	5	5	0,75	0,75
Итого	1	24	24	4,25	3,85

Исходя из данных таблицы 1 можно сделать вывод, что технология «А» имеет преимущества над технологией «Б».

Данная технология дает пользователю гораздо больше возможностей для анализа получаемых данных о сварочном процессе, а также намного более выгодна с финансовой точки зрения, что может сделать ее более предпочтительным вариантом.

3.1.3 SWOT-анализ

SWOT анализ – это определение сильных и слабых сторон проекта, выявление возможностей и угроз по его осуществлению. Этот анализ проводят для выявления внешней и внутренней среды проекта. Проводится этот анализ в три этапа.

Первый этап.

Данный этап заключается в описании сильных и слабых сторон проекта, в выявлении возможностей и угроз для реализации проекта, которые проявились или могут появиться в его внешней среде.

Сильные стороны проекта – это его факторы, которые характеризуют конкурентоспособную сторону научно–исследовательского проекта. Сильные стороны свидетельствуют о том, что у проекта есть отличительное преимущество или особые ресурсы, являющиеся особенными с точки зрения конкуренции.

К сильным сторонам проекта относятся:

1. Объем предоставляемых для анализа данных – С1.
2. Гибкость за счет возможности подключения стороннего инструмента для анализа – С2.
3. Малая стоимость итогового проектного решения – С3.

К слабым сторонам относятся:

1. Проект находится на стадии прототипа – Сл1.
2. В дополнение к программному обеспечению не предоставляется оборудование – Сл2.

К возможностям проекта относятся:

1. Слабая распространенность инструментов-конкурентов в России – В1.

2. Привлечение внимания инвесторов – В2.

К угрозам проекта относятся:

1. Неудовлетворенность продуктом в связи с узкой направленностью – У1.

2. Незаинтересованность в продукте – У2.

Второй этап.

Данный этап состоит в выявлении соответствия сильных и слабых сторон научно-исследовательского проекта внешним условиям окружающей среды. Это соответствие или несоответствие должны помочь выявить степень необходимости проведения стратегических изменений.

Таблица 2 – Интерактивная матрица проекта

Сильные стороны				
		С1	С2	С3
Возможности	В1	+	+	+
	В2	+	+	+
Угрозы	У1	-	-	+
	У2	-	-	-
Слабые стороны				
		Сл1		Сл2
Возможности	В1	+		-
	В2	+		+
Угрозы	У1	+		-
	У2	+		-

Третий этап.

В рамках третьего этапа составляется итоговая матрица SWOT-анализа.

Таблица 3 – Итоговая матрица SWOT-анализа.

	Сильные стороны: С1. Объем предоставляемых для анализа данных; С2. Гибкость за счет возможности подключения стороннего инструмента для анализа;	Слабые стороны: Сл1. Проект находится на стадии прототипа; С2. В дополнение к программному обеспечению не предоставляется оборудование.

	С3. Малая стоимость итогового проектного решения.	
Возможности: В1. Слабая распространенность инструментов-конкурентов в России – В1; В2. Привлечение внимания инвесторов;	Обширные возможности в области анализа могут привлечь как инвесторов, так и клиентов, а благодаря низкому уровню осведомленности о конкурирующих разработках, можно надежно закрепиться на отечественном рынке.	Возможно, что нахождение продукта в самом начале процесса разработки отпугнет потенциального инвестора или покупателя, также как и необходимость в приобретении дополнительного оборудования в виде осциллографов.
Угрозы: У1. Неудовлетворенность продуктом в связи с узкой направленностью; У2. Незаинтересованность в продукте.	Низкая стоимость продукта может компенсировать неудовлетворенность, а незаинтересованность в продукте не критична поскольку представленный продукт лишь студенческая наработка.	Нахождение продукта на стадии прототипа возможно еще больше отпугнет потенциального покупателя, однако может снизить недовольство в случае приобретения, поскольку у продукта есть перспективы на улучшение.

В ходе анализа были рассмотрены все сильные и слабые стороны научного проекта, а также разъяснены все его последствия. Как позитивные, так и негативные – возможности и угрозы. SWOT-анализ показал, что данная разработка может иметь два кардинально отличающихся исхода событий. Первый это очень хороший вариант, при котором к данной разработке появится интерес у сторонних производителей, исследователей. Что приведет к увеличению финансирования и появлению новых кадров, которые ускорят разработку либо при помощи своей высокой квалификации, либо простым увеличением количеством исполнителей, что позволит более быстро проводить некоторые этапы разработки. Второй вариант это тот при котором исследование не найдет должного финансирования, и будет либо закрыто, либо будет двигаться к конечному этапу с сильным промедлением.

3.2 Планирование научно-исследовательских работ

Планирование комплекса научно-исследовательских работ осуществляется в порядке:

1. Определение структуры работ в рамках научного исследования.
2. Определение количества исполнителей для каждой из работ.
3. Установление продолжительности работ.
4. Построение графика проведения научных исследований.

Для выполнения научных исследований формируется рабочая группа, в состав которой могут входить научные сотрудники и преподаватели, инженеры, техники и лаборанты, численность групп может варьироваться. По каждому виду запланированных работ устанавливается соответствующая должность исполнителей.

Работу выполняло 2 человека: Гордынец А.С. – руководитель (Р), Иванов В.Н. – инженер (И).

Трудоемкость выполнения НИР оценивается экспертным путем в человеко-днях и носит вероятностный характер, т.к. зависит от множества трудно учитываемых факторов. Разделим выполнение дипломной работы на этапы, представленные в таблице 4.

Таблица 4 – Перечень этапов, работ и распределение ролей.

Основные этапы	№ раб	Содержание работ	Должность исполнителя
Разработка темы диссертации	1	Составление и утверждение темы диссертации, утверждение плана-графика.	Научный руководитель
Подготовка к выполнению работ	2	Календарное планирование работ	инженер
	3	Выбор инструментов, технологий и подходов к разработке	инженер
Теоретические исследования	4	Изучение документации к используемым инструментам и оборудованию	инженер
Разработка прототипа	5	Разработка ядра приложения	инженер
	6	Разработка модуля вспомогательных функций	инженер
	7	Разработка графического интерфейса	инженер
	8	Разработка модуля взаимодействия с базой данных	инженер

	9	Разработка модуля взаимодействия с контролирующим оборудованием	инженер
	10	Разработка модуля интеграции сторонней логики в приложение	инженер
Тестирование разработанной программы	11	Тестирование с использованием наборов сгенерированных случайным образом данных	инженер
	12	Тестирование с использованием наборов данных, полученных с реальных сварочных процессов	инженер
Оформление отчета по НИР	13	Составление пояснительной записки	инженер

3.2.1 Определение трудоемкости выполнения работ

Трудовые затраты в большинстве случаях образуют основную часть стоимости разработки, поэтому важным моментом является определение трудоемкости работ каждого из участников научного исследования.

Трудоемкость выполнения научного исследования оценивается экспертным путем в человеко-днях и носит вероятностный характер, т.к. зависит от множества трудно учитываемых факторов. Для определения ожидаемого (среднего) значения трудоемкости используется формула (2):

$$t_{\text{ож}i} = \frac{3t_{min\,i} + 2t_{max\,i}}{5}, \qquad (2)$$

где $t_{\text{ож}i}$ – ожидаемая трудоемкость выполнения i-ой работы чел.-дн.;

t_{mini} – минимально возможная трудоемкость выполнения заданной i-ой работы (оптимистическая оценка: в предположении наиболее благоприятного стечения обстоятельств), чел.-дн.;

t_{maxi} – максимально возможная трудоемкость выполнения заданной i-ой работы (пессимистическая оценка: в предположении наиболее неблагоприятного стечения обстоятельств), чел.-дн.

Исходя из ожидаемой трудоемкости работ, определяется продолжительность каждой работы в рабочих днях ТР, учитывающая параллельность выполнения работ несколькими исполнителями. Такое вычисление необходимо для обоснованного расчета заработной платы, так как удельный вес зарплаты в общей сметной стоимости научных исследований составляет около 65 %.

$$T_{Pi} = \frac{t_{\text{ож}i}}{\text{Ч}_i}, \tag{3}$$

где T_{Pi} – продолжительность одной работы, раб. дн.;

$t_{ожі}$ – ожидаемая трудоемкость выполнения одной работы, чел.-дн.;

Ч_i – численность исполнителей, выполняющих одновременно одну и ту же работу на данном этапе, чел.

3.2.2 Разработка графика проведения научного исследования

Диаграмма Ганта – горизонтальный ленточный график, на котором работы по теме представляются протяженными во времени отрезками, характеризующимися датами начала и окончания выполнения данных работ.

Для удобства построения графика, длительность каждого из этапов работ из рабочих дней следует перевести в календарные дни. Для этого необходимо воспользоваться формулой (4):

$$T_{Ki} = T_{Pi} \cdot k_{\text{кал}}, \tag{4}$$

где $T_{кі}$ – продолжительность выполнения *i*-й работы в календарных днях;

$T_{рі}$ – продолжительность выполнения *i*-й работы в рабочих днях;

$k_{\text{кал}}$ – коэффициент календарности.

Коэффициент календарности определяется по формуле (5):

$$k_{\text{кал}} = \frac{T_{\text{кал}}}{T_{\text{кал}} - T_{\text{вых}} - T_{\text{пр}}}, \tag{5}$$

где $T_{кал}$ – количество календарных дней в году;

$T_{вых}$ – количество выходных дней в году;

$T_{пр}$ – количество праздничных дней в году.

Таким образом:

$$k_{кал} = \frac{365}{365 - 52 - 14} = 1{,}22$$

Рассчитанные значения в календарных днях по каждой работе T_{ki} необходимо округлить до целого числа.

Все рассчитанные значения представлены в таблице 5.

Таблица 5 – Временные показатели проведения научного исследования

Название работы	Трудоемкость работ						Длительность работы в рабочих днях, Т	Длительность работы в календарных днях, Т
	t_{mini}, чел.-дни		t_{maxi}, чел.-дни		$t_{ожi}$, чел.-дни			
	Исп. 1	Исп. 2	Исп. 1	Исп. 2	Исп. 1	Исп. 2		
1	2	3	4	5	6	7	8	9
1. Составление и утверждение темы диссертации, утверждение плана-графика.	1	-	2	-	1,4	-	1,4	2
2. Календарное планирование работ.	-	2	-	3	-	2,4	2,4	3
3. Выбор инструментов, технологий и подходов к разработке.	-	2	-	3	-	2,4	2,4	3
4. Изучение документации к используемым инструментам и оборудованию.	-	2	-	3	-	2,4	2,4	3
5. Разработка ядра приложения.	-	10	-	20	-	14	14	17
6. Разработка модуля вспомогательных функций.	-	10	-	20	-	14	14	17
7. Разработка графического интерфейса.	-	10	-	20	-	14	14	17

8. Разработка модуля взаимодействия с базой данных.	-	10	-	20	-	14	14	17
9. Разработка модуля взаимодействия с контролирующим оборудованием.	-	10	-	20	-	14	14	17
10. Разработка модуля интеграции сторонней логики в приложение.	-	10	-	20	-	14	14	17
11. Тестирование с использованием наборов сгенерированных случайным образом данных.	-	1	-	2	-	1,4	1,4	2
12. Тестирование с использованием наборов данных, полученных с реальных сварочных процессов.	-	2	-	3	-	2,4	2,4	3
13. Составление пояснительной записки.	-	1	-	2	-	1,4	28	34
Итого	1	70	2	136	1,4	96,4	124,4	152

Примечание: Исп.1 – научный руководитель; Исп.2 – инженер.

На основе таблицы составляется календарный план-график выполнения проекта с использованием диаграммы Ганта (рисунок 24).

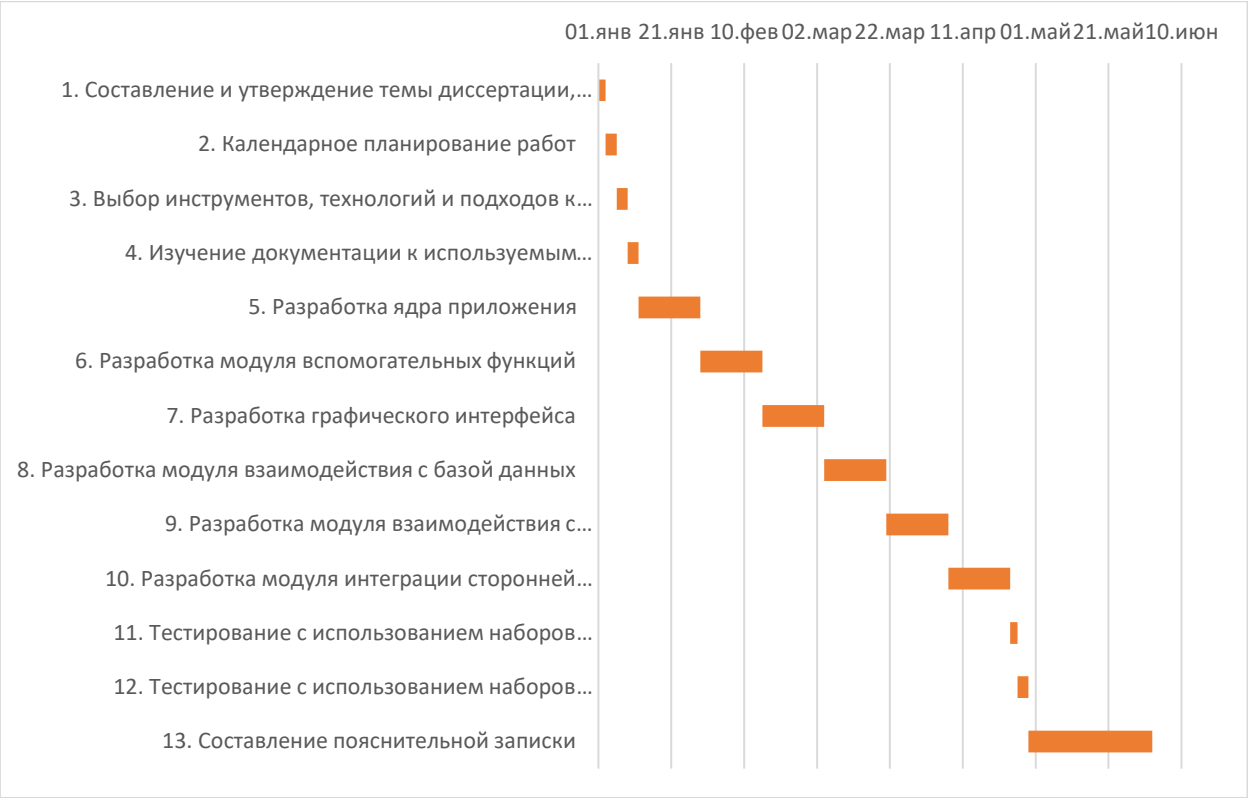


Рисунок 31 – Диаграмма Ганта

3.3 Бюджет научно-технического исследования (НТИ)

При планировании бюджета НТИ должно быть обеспечено полное и достоверное отражение всех видов расходов, связанных с его выполнением. В процессе формирования бюджета НТИ используется следующая группировка затрат по статьям:

1. Материальные затраты НТИ.
2. Затраты на специальное оборудование для научных(экспериментальных) работ.
3. Основная заработная плата исполнителей темы.
4. Дополнительная заработная плата исполнителей темы.
5. Отчисления во внебюджетные фонды (страховые отчисления).
6. Затраты на научные и производственные командировки.
7. Контрагентые расходы.
8. Накладные расходы.

3.3.1 Расчет материальных затрат

Данная статья включает стоимость всех материалов, используемых при разработке проекта:

– Приобретаемые со стороны сырье и материалы, необходимые для создания научно-технической продукции.

– Покупные материалы, используемые в процессе создания научно-технической продукции для обеспечения нормального технологического процесса и для упаковки продукции или расходуемых на другие производственные и хозяйственные нужды (проведение испытаний, контроль, содержание, ремонт и эксплуатация оборудования, зданий, сооружений, других основных средств и прочее), а также запасные части для ремонта оборудования, износа инструментов, приспособлений, инвентаря, приборов, лабораторного оборудования и других средств труда, не относимых к основным средствам, износ спецодежды и других малоценных и быстроизнашивающихся предметов.

– Покупные комплектующие изделия и полуфабрикаты, подвергающиеся в дальнейшем монтажу или дополнительной обработке.

– Сырье и материалы, покупные комплектующие изделия и полуфабрикаты, используемые в качестве объектов исследований (испытаний) и для эксплуатации, технического обслуживания и ремонта изделий – объектов испытаний (исследований).

Расчет материальных затрат осуществляется по следующей формуле (6):

$$З_м = (1 + k_т) \sum_{i=1}^{m} Ц_i \cdot N_{расх\,i}\,, \qquad (6)$$

где m – количество видов материальных ресурсов, потребляемых при выполнении научного исследования;

$N_{расхi}$ – количество материальных ресурсов i-го вида, планируемых к использованию при выполнении научного исследования (шт., кг, м, м^2 и т.д.);

$Ц_i$ – цена приобретения единицы *i*-го вида потребляемых материальных ресурсов (руб./шт., руб./кг, руб./м, руб./м^2 и т.д.);

$k_Т$ – коэффициент, учитывающий транспортно-заготовительные расходы.

Величина коэффициента ($k_Т$), отражающего соотношение затрат по доставке материальных ресурсов и цен на их приобретение, зависит от условий договоров поставки, видов материальных ресурсов, территориальной удаленности поставщиков и т.д. Транспортные расходы принимаются в пределах 15-25% от стоимости материалов. Материальные затраты, необходимые для данной разработки, представлены в таблице 6.

Таблица 6 – Материальные затраты

Наименованиематериала	**Единица измерения**	**Количество**	**Цена за ед.,** руб.	**Затраты на материалы,** руб.
Белая бумага	Пачка	1	135	135
Ручка	Шт.	2	30	60
Сварочная проволока Св-08Г2С	Кг	5	160	1000
Газовая смесь 80% аргона и 20% углекислого газа	Баллон	1	1200	1500
Сварочный костюм спилковый Gigant GT-109	Шт.	1	3890	4862,5
Итого				**7 557,5**

3.3.2 Расчет затрат на специальное оборудование для проведения научных (экспериментальных) работ

В данную статью включают все затраты, связанные с приобретением специального оборудования (приборов, контрольно-измерительной аппаратуры, стендов, устройств и механизмов), необходимого для проведения работ по конкретной теме. Определение стоимости спецоборудования производится по действующим прейскурантам, а в ряде случаев по договорной цене.

При приобретении спецоборудования необходимо учесть затраты по его доставке и монтажу в размере 15% от его цены. Стоимость оборудования,

используемого при выполнении конкретного НТИ и имеющегося в данной научно-технической организации, учитывается в калькуляции в виде амортизационных отчислений. Результаты расчетов представлены в таблице.

Для проведения научно-исследовательской работы требуется: персональный компьютер, осциллограф Rigol DS1054Z, Цифровой импульсный сварочный полуавтомат ALLOY MC-351MX PULSE, Редуктор универсальный Сварог Tech Control UNI, Маска сварочная AURORA Sun7.

Затраты на спецоборудование для научных работ приведены в таблице 7.

Таблица 7 – Расчет бюджета затрат на приобретение спецоборудования для научных работ

Наименование оборудования	Цена, руб	Цена с учетом затрат на доставку и монтаж, руб
Цифровой импульсный сварочный полуавтомат ALLOY MC-351MX PULSE	262000	301300
Редуктор универсальный Сварог Tech Control UNI	2922	3360,3
Маска сварочная AURORA Sun7	4600	5290
Персональный компьютер	36990	42538,5
осциллограф Rigol DS1054Z	54372	62527,8
Итого:	**460277**	**415016,6**

3.3.3 Основная заработная плата исполнителей

Величина расходов по заработной плате определяется исходя из трудоемкости выполняемых работ и действующей системы окладов и тарифных ставок. Основная заработная плата научного руководителя и консультантов рассчитывается на основании отраслевой оплаты труда. Отраслевая система оплаты труда в ТПУ предполагает следующий состав заработной платы:

1. Оклад – определяется предприятием. В ТПУ оклады распределены в соответствии с занимаемыми должностями, например, ассистент, ст. преподаватель, доцент, профессор.

2. Стимулирующие выплаты – устанавливаются руководителем подразделений за эффективный труд, выполнение дополнительных обязанностей и т.д.

3. Иные выплаты; районный коэффициент.

Месячный должностной оклад работника:

$$З_м = З_{тс} \cdot (1 + k_{пр} + k_д) \cdot k_р, \quad (7)$$

где $З_{тс}$ – заработная плата по тарифной ставке ТПУ, руб.;

$k_{пр}$ – премиальный коэффициент, равный 0,3 (т.е. 30% от $З_{тс}$);

$k_д$ – коэффициент доплат и надбавок составляет примерно 0,2-0,5 (в НИИ и на промышленных предприятиях – за расширение сфер обслуживания, за профессиональное мастерство, за вредные условия: 15-20 % от $З_{тс}$);

$k_р$ – районный коэффициент, равный 1,3 (для Томска).

Среднедневная заработная плата рассчитывается по формуле:

$$З_{дн} = \frac{З_м \cdot M}{F_д}, \quad (8)$$

где $З_м$ – месячный должностной оклад работника, руб.;

M – количество месяцев работы без отпуска в течение года: при отпуске в 48 раб. дней M = 10,4 месяца, 6-дневная неделя;

$F_д$ – действительный годовой фонд рабочего времени научно-технического персонала, раб. дн. (таблица 8).

Таблица 8 – Баланс рабочего времени

Показатели рабочего времени	**Научный руководитель**	**Бакалавр**
Календарное число дней	365	365
Количество нерабочих дней - выходные дни - праздничные дни	 52 14	 52 14
Потери рабочего времени		

- отпуск - невыходы по болезни	48 7	48 7
Действительный годовой фонд рабочего времени	245	245

Расчет основной заработной платы представлен в таблице 9.

Таблица 9 – Расчет основной заработной платы

Категория	$З_{тс}$, руб.	$k_{пр}$	$k_{р}$	$З_{м}$, руб.	$З_{дн}$, руб.	$Т_{р}$, раб.дн.	$З_{осн}$, руб.
Научный руководитель							
ППС3	35120	0,3	1,3	59352,8	2519,5	2	5039
Бакалавр							
ППС1	8600	0,3	1,3	14534,0	616,9	150	92535
Итого							**97574**

3.3.4 Дополнительная заработная плата исполнителей

Затраты по дополнительной заработной плате исполнителей темы учитывают величину предусмотренных Трудовым кодексом РФ доплат за отклонение от нормальных условий труда, а также выплат, связанных с обеспечением гарантий и компенсаций. Расчет дополнительной заработной платы ведется по следующей формуле:

$$З_{доп} = k_{доп} \cdot З_{осн}, \quad (9)$$

где $k_{доп}$ – коэффициент дополнительной заработной платы (на стадии проектирования принимается равным 0,12 – 0,15). Примем $k_{доп}$ = 0, 15.

Общая заработная плата исполнителей представлена в таблице 10.

Таблица 10 – Общая заработная плата исполнителей

Исполнитель	$З_{осн}$, руб.	$З_{доп}$, руб.	$З_{зп}$, руб.
Научный руководитель	5039	755,85	5794,85
Бакалавр	92535	13880,25	106415,25
Итого	**97574**	**14636,1**	**112210,1**

3.3.5 Отчисления во внебюджетные фонды

В данной статье расходов отражаются обязательные отчисления по установленным законодательством Российской Федерации нормам органам государственного социального страхования (ФСС), пенсионного фонда (ПФ) и медицинского страхования (ФФОМС) от затрат на оплату труда работников.

Величина отчислений во внебюджетные фонды определяется исходя из следующей формулы:

$$З_{внеб} = k_{внеб} \cdot (З_{осн} + З_{доп}), \quad (10)$$

где $k_{внеб}$ – коэффициент отчислений на уплату во внебюджетные фонды (пенсионный фонд, фонд обязательного медицинского страхования и пр.).

Общая ставка взносов составляет в 2021 году –30% (ст. 425 НК РФ). Для сотрудников Томского Политехнического Университета этот коэффициент составляет 30,2%.

В таблице 11 представлены отчисления во внебюджетные фонды.

Таблица 11 – Отчисления во внебюджетные фонды

Исполнитель	Основная заработная плата, руб.	Дополнительная заработная плата, руб.
Научный руководитель	5039	1521,78
Бакалавр	92535	27945,57
Коэффициент отчислений во внебюджетные фонды	0,302	
Итого:	**29467,35**	

3.3.6 Накладные расходы

Накладные расходы учитывают прочие затраты, не попавшие в предыдущие статьи расходов: печать и ксерокопирование материалов исследования, оплата услуг связи, электроэнергии, почтовые и телеграфные расходы, размножение материалов и т.д.

Результаты расчета накладных расходов представлены в таблице 12:

Таблица 12 – Расчет накладных расходов

Статья расходов	Единица измерения	Количество	Цена за единицу, руб.	Затраты по статье $З_{накл.ст}$, руб.
Электроэнергия	кВт · час	296,54	2,73	809,55
Печать	Страницы	102	7	714
Интернет	Дни	152	12	1824
Итого:				3347,55

3.3.7 Формирование бюджета научно-исследовательской работы

Рассчитанная величина затрат научно-исследовательской работы является основой для формирования бюджета затрат проекта, который при формировании договора с заказчиком защищается научной организацией в качестве нижнего предела затрат на разработку научно-технической продукции.

В таблице 13 представлен расчет бюджета НТИ.

Таблица 13 – Расчет бюджета затрат НТИ

Наименование статьи	Сумма, руб.	Сумма, %
1. Материальные затраты	7557,5	1,33%
2. Затраты на специальное оборудование для научных (экспериментальных) работ	415016,6	73,12%
3. Затраты по основной заработной плате исполнителей темы	97574	17,19%
4. Затраты по дополнительной заработной плате исполнителей темы	14636,1	2,58%
5. Отчисления во внебюджетные фонды	29467,35	5,19%
6. Накладные расходы	3347,55	0,59%
Бюджет затрат НТИ	567599,1	100%

3.4 Определение ресурсной (ресурсосберегающей), финансовой, бюджетной, социальной и экономической эффективности

Определение эффективности происходит на основе расчета интегрального показателя эффективности научного исследования. Его нахождение связано с определением финансовой эффективности и ресурсоэффективности в ходе оценки бюджета затрат трех (или более) вариантов исполнения научного исследования. Для этого наибольший интегральный показатель реализации технической задачи принимается за базу расчета (как знаменатель), с которым соотносится финансовые значения по всем вариантам исполнения.

Интегральный финансовый показатель разработки определяется как:

$$I_{\text{финр}}^{\text{исп } i} = \frac{\Phi_{\text{p}i}}{\Phi_{max}}, \qquad (11)$$

где $I_{\text{финр}}^{\text{исп } i}$ – интегральный финансовый показатель разработки;

$\Phi_{\text{p}i}$ – стоимость i-го варианта исполнения;

Φ_{max} – максимальная стоимость исполнения научно-исследовательского проекта (в т.ч. аналоги).

Интегральный показатель ресурсоэффективности вариантов исполнения объекта исследования можно определить следующим образом:

$$I_{\text{p}i} = \sum a_i \cdot b_i\,, \qquad (12)$$

где I_{pi}– интегральный показатель ресурсоэффективности для i-го варианта исполнения разработки;

a_{i} – весовой коэффициент i-го варианта исполнения разработки;

b_{i}^{a}, b_{i}^{p} – бальная оценка i-го варианта исполнения разработки, устанавливается экспертным путем по выбранной шкале оценивания;

n – число параметров сравнения.

В таблице 14 показана сравнительная оценка характеристик вариантов исполнения проекта.

Таблица 14 – Сравнительная оценка характеристик вариантов исполнения проекта

Объект исследования / Критерии	Весовой коэффициент параметра	А	Б
1. Простота использования.	0,15	2	4
2. Подробность предоставляемых данных.	0,25	4	4
3. Уровень нагрузки на оборудование.	0,2	3	3
4. Разнообразие доступных возможностей анализа.	0,25	5	3
5. Надежность.	0,15	4	4
ИТОГО	1		
$I_{\text{р}}$		3,75	3,55

Интегральный показатель эффективности вариантов исполнения разработки определяется на основании интегрального показателя ресурсоэффективности и интегрального финансового показателя по формуле:

$$I_{\text{исп } i} = \frac{I_{\text{р } i}}{I_{\text{финр}}^{\text{исп } i}}, \quad (13)$$

Сравнение интегрального показателя эффективности вариантов исполнения разработки позволит определить сравнительную эффективность проекта и выбрать наиболее целесообразный вариант из предложенных. Сравнительная эффективность проекта ($Э_{ср}$):

$$Э_{\text{ср}} = \frac{I_{\text{исп 1}}}{I_{\text{исп 2}}}, \quad (14)$$

Сравнительная эффективность разработки показана в таблице 15.

Таблица 15 – Сравнительная эффективность разработки

№	Показатели	А	Б

п/п			
1	Интегральный финансовый показатель разработки	1	1
2	Интегральный показатель ресурсоэффективности разработки	3,75	3,55
3	Интегральный показатель эффективности	3,75	3,55
4	Сравнительная эффективность вариантов исполнения	1	0,95

Сравнительный анализ интегральных показателей эффективности показывает, что предпочтительным для рассмотрения является вариант исполнения «А», так как данный вариант является наиболее экономичным и ресурсоэффективным.

Выводы

В рамках данного раздела ВКР была определена конкурентоспособность способа механизированной сварки с применением модуляции сварочного тока. Был составлен перечень этапов работ и определена их трудоемкость, построен календарный план-график выполнения работ. Рассчитан ориентировочный бюджет на исследование научной разработки, а также проведена оценка эффективности научного исследования с позиции ресурсосбережения и сравнительная эффективность разработки.

ЗАДАНИЕ ДЛЯ РАЗДЕЛА
«СОЦИАЛЬНАЯ ОТВЕТСТВЕННОСТЬ»

Обучающемуся:

Группа	ФИО
1ВМ11	Иванов Владимир Николаевич

Школа	ИШНКБ	Отделение (НОЦ)	ОЭИ
Уровень образования	Магистратура	Направление/специальность	15.04.01 Машиностроение/ Машины и технологии сварочного производства

Исходные данные к разделу «Социальная ответственность»:	
Характеристика объекта исследования (вещество, материал, прибор, алгоритм, методика) и области его применения. Описание рабочей зоны (рабочего места) при разработке проектного решения/при эксплуатации	*Объект исследования:* разработка программы для анализа сварочного процесса *Область применения:* контроль качества выполнения работ сварки плавлением, автоматизация контроля качества *Рабочая зона:* офис *Размеры помещения:* 5*6 м *Количество и наименование оборудования рабочей зоны:* ноутбук, осциллограф *Рабочие процессы, связанные с объектом исследования, осуществляющиеся в рабочей зоне:* съем тестовых данных с осциллографа
Перечень вопросов, подлежащих исследованию, проектированию и разработке:	
1. Правовые и организационные вопросы обеспечения безопасности <u>при разработке проектного решения:</u> – специальные (характерные при эксплуатации объекта исследования, проектируемой рабочей зоны) правовые нормы трудового законодательства; – организационные мероприятия при компоновке рабочей зоны.	1. ГОСТ 12.2.032-78 Рабочее место при выполнении работ сидя; 2. ГОСТ 22269-76 Рабочее место оператора. Взаимное расположение элементов рабочего места; 3. СП 2.2.3670-20 Санитарно-эпидемиологические требования к условиям труда; 4. Федеральный закон от 28.12.2013 №426-ФЗ «О специальной оценке условий труда»; 5. Трудовой кодекс Российской Федерации №197-ФЗ от 30.12.2001. 6. СанПиН 1.2.3685-21 "Гигиенические нормативы и требования к обеспечению безопасности и (или) безвредности для человека факторов среды обитания";
2. Производственная безопасность <u>при разработке проектного решения</u>: – Анализ выявленных вредных и опасных производственных факторов – Расчет уровня опасного или вредного производственного фактора	**Вредные факторы:** 1. производственные факторы, связанные со световой средой и характеризуемые чрезмерными характеристиками световой среды, затрудняющими безопасное ведение трудовой и производственной деятельности, а именно: отсутствие или недостаток необходимого естественного освещения, отсутствие или недостатки необходимого искусственного освещения; 2. Монотонность труда, вызывающая монотонию; 3. Длительное сосредоточенное наблюдение. **Опасные факторы:** 1. Производственные факторы, связанные с электрическим током, вызываемым разницей электрических потенциалов, под действие которого попадает работающий, включая действие молнии и высоковольтного разряда в виде дуги, а также электрического разряда живых организмов; 2. Повышенное образование электростатических разрядов.

	Требуемые средства коллективной и индивидуальной защиты от выявленных факторов: 1. защита от поражения электрическим током; 2. инструктаж рабочих по работе с оборудованием; **Расчет:** расчет системы искусственного освещения.
3. Экологическая безопасность при эксплуатации	Воздействие на атмосферу: загрязнение воздуха выбросами газа и дыма при сборе данных для анализа сварочного процесса. Воздействие на литосферу: наличие отходов, состоящих из металлической стружки, окалины и шлака.
4. Безопасность в чрезвычайных ситуациях при эксплуатации	Возможные ЧС: 1. возникновение пожара, взрыва; Наиболее типичная ЧС: возникновение пожара, взрыва.
Дата выдачи задания для раздела по линейному графику	

Задание выдал консультант по разделу «Социальная ответственность»:

Должность	ФИО	Ученая степень, звание	Подпись	Дата
Доцент ООД	Антоневич Ольга Алексеевна	к.б.н.		

Задание принял к исполнению обучающийся:

Группа	ФИО	Подпись	Дата
1ВМ11	Иванов Владимир Николаевич		

4 СОЦИАЛЬНАЯ ОТВЕТСТВЕННОСТЬ

В разделе «Социальная ответственность» анализируются вредные и опасные факторы, которые могут возникнуть при разработке и проведении испытаний, рассматриваются вопросы экологической безопасности и безопасности в случае возникновения чрезвычайной ситуации. Целью данного раздела является минимизация несчастных случаев и вредного воздействия на окружающую среду при эксплуатации проектного решения.

Объектом исследования в работе является разработка программы для анализа сварочного процесса. Основной задачей сварочного процесса является получение неразъёмных соединений по свойствам подобных цельным материалам. Многие компании предпринимают попытки усовершенствовать этот процесс. Тем же целям служит разрабатываемая программа.

Данное проектное решение может использоваться, как в лабораторных условиях, так и на производстве в тех областях, где качество сварных конструкций является первостепенным. Среди таких областей применения наиболее распространенными являются машиностроение и неразрушающий контроль.

В качестве рабочей зоны выступает офисное помещение размером 16 м^2, оснащенная персональным компьютером, осциллографом и программным комплексом Visual Studio 2022 Community для разработки программного обеспечения.

Основными вредными факторами при проведении работ являются неудовлетворительные показатели микроклимата, освещения, а также длительное нахождение в рабочей позе и умственное перенапряжение. Кроме этого, существует опасность поражения электрическим током и возникновения пожара в помещении.

4.1 Правовые и организационные вопросы обеспечения безопасности

Согласно статье 216 Трудового Кодекса РФ [12] каждый работник имеет право на:

– рабочее место, соответствующее требованиям охраны труда;

– обязательное социальное страхование от несчастных случаев на производстве и профессиональных заболеваний;

– отказ от выполнения работ в случае возникновения опасности для его жизни и здоровья вследствие нарушения требований охраны труда до устранения такой опасности;

– обеспечение в соответствии с требованиями охраны труда за счет средств работодателя средствами коллективной и индивидуальной защиты и смывающими средствами;

– обязанности по обеспечению безопасных условий и охраны труда возлагаются на работодателя [ТК РФ Статья 214].

Кроме того, работникам должны обеспечиваться следующие условия:

– нормальная продолжительность рабочего времени, которая не может превышать 40 часов в неделю [ТК РФ Статья 91];

– работодатели, работники и их представители должны совместно вырабатывать меры защиты персональных данных работников [ТК РФ Статья 86].

Рабочей зоной при разработке проектного решения является офисное помещение. Согласно ФЗ «О специальной оценке условий труда» [13] работа в офисе имеет оптимальные условия (1 класс). При данном классе условий труда воздействие на работника вредных и опасных производственных факторов не превышает уровни, установленные нормативами условий труда и принятые в качестве безопасных для человека, и создаются предпосылки для поддержания высокого уровня работоспособности работника.

Основным рабочим местом при разработке программного обеспечения и его тестирования является рабочий стол, на котором расположено все необходимое специальное оборудования для работы. Согласно требованиям, изложенным в ГОСТ 12.2.032-78 ССБТ [14], ГОСТ 22269-76 [15], ГОСТ Р 50923-96 [16] и СП 2.2.3670-20 [17], рабочее место и работа с ПК должны удовлетворять следующим положениям:

– производственное оборудование и рабочие столы должны иметь пространство для размещения ног высотой не менее 600 мм, глубиной – не менее 450 мм на уровне колен и 600 мм на уровне стоп, шириной не менее 500 мм;

– конструкция рабочего места и взаимное расположение всех его элементов (сиденье, органы управления, средства отображения информации и т.д.) должны соответствовать антропометрическим, физиологическим и психологическим требованиям, а также характеру работы;

– конструкция рабочего стула или кресла должны обеспечивать поддержание рациональной рабочей позы при работе с ПК;

– взаимное расположение элементов рабочего места должно обеспечивать возможность осуществления всех необходимых движений и перемещений для эксплуатации технического обслуживания оборудования;

– наиболее важные и часто используемые органы управления должны быть расположены в зоне досягаемости моторного поля.

Таким образом, конструкция рабочего места и взаимное расположение всех его элементов (сиденье, органы управления, средства отображения информации и т.д.) должны соответствовать антропометрическим, физиологическим и психологическим требованиям, а также характеру работы.

4.2 Производственная безопасность

4.2.1 Анализ опасных и вредных производственных факторов

Опираясь на ГОСТ 12.0.003-2015 [18], проведем анализ опасных и вредных факторов, которые могут возникнуть при разработке проектируемого решения. Результат представлен в таблице 16.

Таблица 16 – Возможные опасные и вредные факторы

Факторы	Нормативные документы
Отсутствие или недостаток необходимого освещения	СП 52.13330.2016. Естественное и искусственное освещение. Актуализированная редакция СНиП 23-05-95[19].
	СанПиН 1.2.3685-21 Гигиенические нормативы и требования к обеспечению безопасности и (или) безвредности для человека факторов среды обитания [20].
Умственное перенапряжение	МР 2.2.9.2311-07 Профилактика стрессового состояния работников при различных видах профессиональной деятельности [21].
Статические физические перегрузки, связанные с рабочей позой	ГОСТ 12.2.032-78 ССБТ. Рабочее место при выполнении работ сидя [14]
Повышенное образование электростатических зарядов	ГОСТ 31610.32-1 – 2015 (IEC/TS 60079-32-1:2013) Взрывоопасные среды. Электростатика. Опасные проявления. Руководство [22].
	ГОСТ 12.1.045-84 ССБТ. Электростатические поля. Допустимые уровни на рабочих местах и требования к проведению контроля [23]
	ГОСТ 12.4.124-83 ССБТ. Средства защиты от статического электричества. Общие технические требования [24].
Электрический ток, вызываемый разницей электрических потенциалов	ГОСТ 12.1.009-2017 ССБТ. Электробезопасность. Термины и определения [25].
	ГОСТ 12.4.011-89 ССБТ. Средства защиты работающих. Общие требования и классификации [26].
	ГОСТ 12.1.038-82 ССБТ. Электробезопасность. Предельно допустимые значения напряжений прикосновения и токов [27]

Далее рассмотрено влияние каждого фактора и обоснование мероприятий по его минимизации.

Отсутствие или недостаток необходимого освещения.

Искусственное освещение в помещениях для эксплуатации ЭВМ должно осуществляться системой общего равномерного освещения. Рабочие столы следует размещать таким образом, чтобы видеодисплейные терминалы были ориентированы боковой стороной к световым проемам, чтобы естественный свет падал преимущественно слева.

В административно-общественных помещениях, в случаях преимущественной работы с документами, следует применять системы комбинированного освещения (к общему освещению дополнительно устанавливаются светильники местного освещения, предназначенные для освещения зоны расположения документов) [19].

Недостаточная освещенность негативно воздействует на зрение, приводит к быстрому утомлению, снижает работоспособность, вызывает дискомфорт.

В офисном помещении освещенность на рабочей зоны от системы искусственного освещения должна составлять порядка 500 Лк. Освещение не должно создавать бликов на поверхности экрана [20].

Освещение на рабочем месте является совмещенным и составляет 530 Лк, что удовлетворяет нормативным требованиям. Рабочий стол обращен боковой стороной к световому проему (окну), присутствует дополнительная лампа для местного освещения.

Умственное перенапряжение

Рабочий процесс подразумевает длительную работу за ПК, а также работу с обширной документацией к используемым инструментам, что вызывает зрительную и умственную нагрузку на организм человека.

Длительное перенапряжение анализаторов может привести к состоянию перенапряжения или утомления работника и, следовательно, к снижению его работоспособности, а также может привести к развитию стресса и невротическому расстройству.

С целью предотвращения риска умственного перенапряжения рекомендуется организовывать режим труда и отдыха. При 8-часовой работе

продолжительность обеденного перерыва должна составлять 30 минут, а регламентированные перерывы рекомендуется устанавливать через 2 часа от начала рабочей смены и через 2 часа после обеденного перерыва продолжительностью 5-7 минут каждый. Во время перерывов целесообразно выполнять комплексы физических упражнений, а в конце рабочего дня рекомендована психологическая разгрузка [21].

Длительное нахождение в рабочей позе

Работа выполняется в положении сидя за рабочим столом, на котором располагается все необходимое оборудование.

Неправильная длительная рабочая поза может привести к болезням, связанным с опорно-двигательной системой. Среди наиболее распространенных заболеваний можно выделить болезни пояснично-крестового и шейно-грудного отдела, а также воспаление органов малого таза в результате недостаточного кровообращения.

При разработке проектного решения работа проводилась на нерегулируемой рабочей поверхности. Исходя из этого числовые значения рабочего места определяются в зависимости от вида работы и высоты сидения. ГОСТ 12.2.032-78 [14] регламентирует при печатании и легких сборочных работах использовать высоту рабочей поверхности 655 мм и высоту сиденья 420 мм. Высота реальной рабочей поверхности составила 660 мм, а высота сиденья 420 мм. Таким образом, делаем вывод, что высота рабочей поверхности удовлетворяет поставленным требованиям.

Профилактика перегрузки из-за длительного нахождения в рабочей позе сводится к рационализации рабочей позы путем совершенствования конструкции рабочего места, производственной гимнастике и смене положения во время регламентированных перерывов.

Повышенное образование электростатических зарядов

Источником возникновения рассматриваемого фактора является электрооборудование, находящееся в рабочей зоне: осциллограф и ПК.

Опасность, вызываемая электростатическим зарядом, включает в себя удар статическим электричеством, приводящим к травме, риск зажигания или взрыва [23].

Допустимые уровни напряженности электростатических полей устанавливаются в зависимости от времени пребывания персонала на рабочих местах. Согласно ГОСТ 12.1.045-84 [23], предельно допустимый уровень напряженности электростатических полей устанавливается равным 60 кВ/м в течение 1 ч. Значение напряженности электростатического поля в офисе не превышает значения 20 кВ/м, следовательно, время пребывания в электростатических полях не регламентируется.

В целях снижения воздействия статического электричества на организм человека предусмотрены следующие средства защиты: заземляющие устройства, увлажняющие устройства, антиэлектростатические вещества (одежда), экранирующие вещества [24].

Электрический ток, вызываемый разницей электрических потенциалов.

Источником электрического тока может служить используемое в рабочей зоне электрооборудование.

При длительном воздействии электрического поля повышается вероятность возникновения заболеваний центральной нервной, сердечно-сосудистой, эндокринной, иммунной систем организма. Поражение электрическим током также может вызывать электрический ожог или более серьезную электротравму [27].

При выборе и расчете технических устройств и других средств защиты учитываются три основных параметра: сила тока, протекающего через тело человека, напряжение прикосновения и длительность протекания (таблица 17) [25].

Таблица 17 – Допустимые значения напряжения прикосновения и силы тока при продолжительности воздействия не более 10 минут в сутки

Род тока	*U*, В	*I*, мА
	не более	
Переменный, 50 Гц	2,0	0,3

Постоянный	8,0	1,0

В качестве средства коллективной и индивидуальной защиты от поражения электрического тока могут быть устройства защитного заземления и зануления, устройства автоматического отключения и знаки безопасности [26].

4.2.2 Расчет системы общего равномерного искусственного освещения

Дано помещение площадью 16 м^2 с длиной A = 6 м, шириной B = 5 м и высотой H = 3,6 м. Необходимо обеспечить освещенность $E_н$ = 500 лк для рабочей поверхности высотой $h_{рп}$ = 0,66 м. Коэффициент запаса $K_з$ = 1,5. Коэффициент неравномерности Z = 1,1.

Расчёт общего равномерного искусственного освещения горизонтальной рабочей поверхности выполняется методом коэффициента светового потока, учитывающим световой поток, отражённый от потолка и стен.

В качестве светильников были выбраны люминесцентные без защитной решетки типа ОД мощностью 40 Вт. Для таких светильников λ = 1,4. Свешивание $h_с$ принято равным 0,6 м. Размещение светильников в помещении определяется следующими параметрами:

Высота светильника над полом: $h_n = H - h_с = 3{,}6 - 0{,}6 = 3$ м.

Расчетная высота: $h = h_n - h_{рп} = 3 - 0{,}66 = 2{,}34$ м < 5 м.

Расстояние между светильниками: $L = \lambda h = 1{,}4 \cdot 2{,}34 = 3{,}276$ м.

Оптимальное расстояние от крайнего ряда светильников до стены:

$l = L/3 = 3{,}276/3 = 1{,}092$ м.

Количество рядов светильников:

$$n_{ряд} = \frac{B - \frac{2}{3}L}{L} + 1 = \frac{5 - 2{,}184}{3{,}276} + 1 = 1{,}86 \approx 2.$$

Количество светильников:

$$n_{\text{св}} = \frac{A - \frac{2}{3}L}{l + 0{,}5} = \frac{6 - 2{,}184}{1{,}592} = 2{,}397 \approx 3.$$

Общее количество светильников: $N = n_{\text{ряд}} n_{\text{св}} = 2 \cdot 3 = 6$.

Общее число ламп в помещении равно $N_{\text{л}}$ = 12, так как в каждом светильнике содержится по две лампы.

Индекс помещения равен: $i = S/(h \cdot (A + B)) = 30/(25{,}74) = 1{,}165$.

Коэффициент отражения стен $\rho_{\text{ст}}$ = 30%, потолка $\rho_{\text{п}}$ = 50%.

Коэффициент использования светового потока: η = 48%.

Световой поток лампы:

$$\Phi = \frac{E_{\text{н}} S K_{\text{з}} Z}{N_{\text{л}} \eta} = \frac{500 \cdot 18 \cdot 1{,}5 \cdot 1{,}1}{12 \cdot 0{,}48} = 2578{,}125 \text{ лм}.$$

Ближайшая стандартная лампа – ЛХБ 40 Вт с потоком 2700 лм.

Проверка полученных значений:

$$-10\% \leq \frac{\Phi_{\text{л.станд}} - \Phi_{\text{л.расч}}}{\Phi_{\text{л.станд}}} \cdot 100\% \leq +20\%;$$

$$-10\% \leq 4{,}515\% \leq +20\%;$$

Электрическая мощность осветительной системы: $P = N_{\text{л}} p_{\text{л}} = 480$ Вт.

На рисунке 12 представлена полученная схема размещения светильников в помещении.

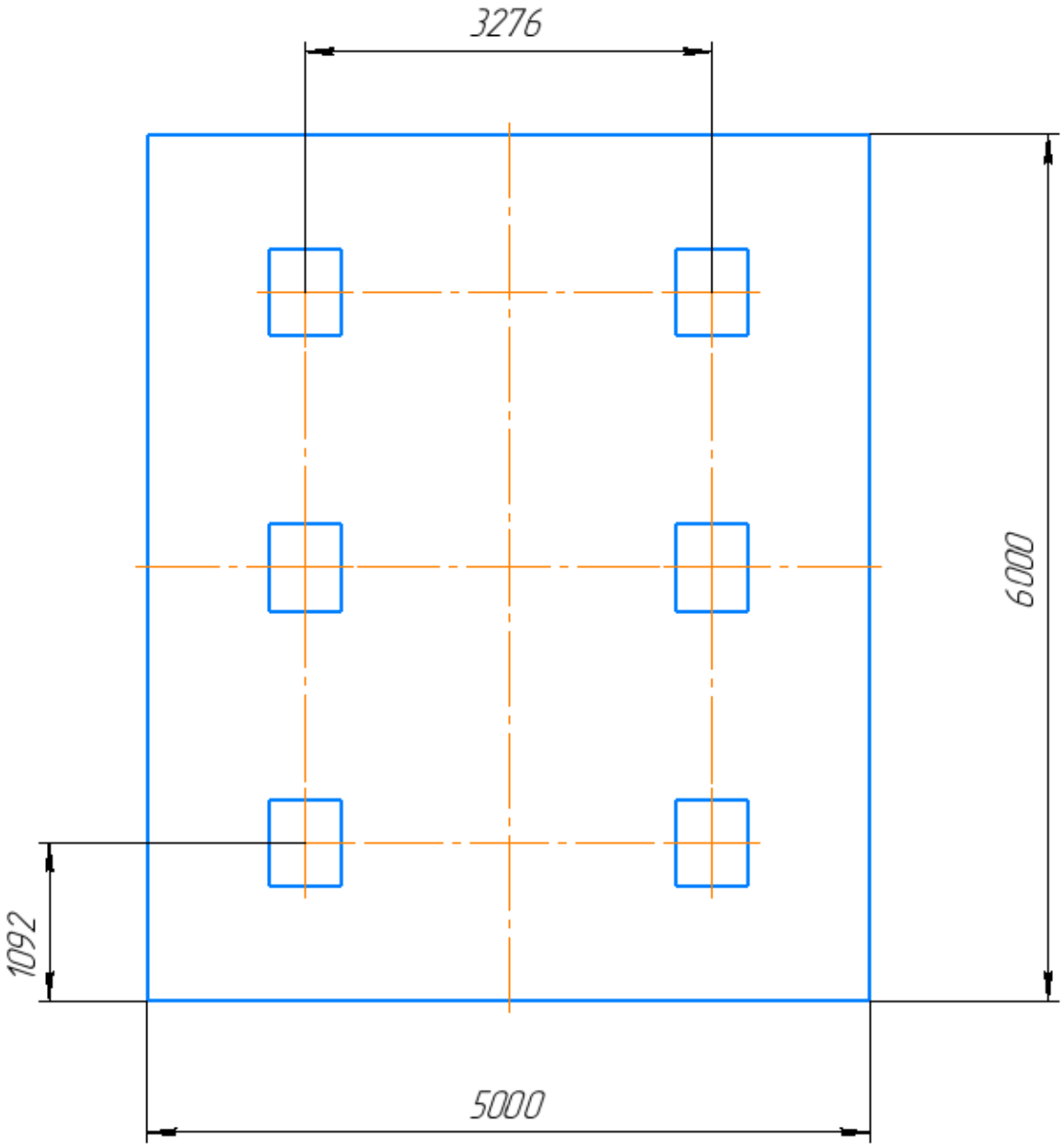


Рисунок 32 – Схема размещения светильников

4.2.3 Экологическая безопасность

Защита атмосферы

Источником загрязнения могут служить выбросы газа и дыма, образующиеся в ходе сварочного процесса. В качестве защитной среды при сварке чаще всего применяют газ аргон и смеси на его основе. Данный газ имеет IV класс опасности по ГОСТ 12.1.007-76 [28].

Основными направлениями мероприятий по охране воздуха могут быть очистка отходящих газообразных выбросов, очистка выбросов в атмосферу через вентиляционные трубы, рассеивание вредных веществ в атмосфере для снижения концентрации ее в приземном слое [29].

Защита литосферы.

Источником загрязнения являются отходы производства и потребления, а именно: металлическая стружка, окалина и шлак, относящиеся к IV классу опасности.

Согласно ГОСТ Р 53692- 2009 [30], не используемые в ближайшей перспективе отходы I-IV класса, а также объекты, не поддающиеся демонтажу с последующей утилизацией, подлежат захоронению или уничтожению.

4.3 Безопасность в чрезвычайных ситуациях

Возможными чрезвычайными ситуациями на рабочем месте могут быть пожар в здании или взрыв электрооборудования.

С учетом специфики работы и наличия электрооборудования в рабочей зоне в ходе эксплуатации проектного решения вероятнее всего возникновение пожара. Среди причин, вызывающих возгорание, можно выделить: токи короткого замыкания; воспламенение измерительного устройства из-за перегрузки; неисправность электрооборудования; сбой в работе электросети.

Согласно ГОСТ 12.1.004-91 [31] для предотвращения возникновения пожара применяются следующие меры: инструктажи по правилам пожарной безопасности; провидение работ только при исправном оборудовании и электропроводке; предохранение электросети от перегрузок. Кроме того, в рабочем помещении должны быть предусмотрены установки пожарной сигнализации, устройства аварийного отключения установок, средства пожаротушения и план эвакуации на случай возникновения ЧС.

В случае возникновения ЧС необходимо сохранять спокойствие и действовать быстро. Следует убедиться, что электрооборудование обесточено и использовать все доступные способы для тушения огня. Если потушить

огонь в кратчайшее время невозможно необходимо вызвать пожарную охрану предприятия. При эвакуации горящие помещения и задымленные места следует проходить быстро, задержав дыхание, защитив нос и рот влажной плотной тканью. После ЧС необходимо следовать инструкциям спасательных подразделений.

Возможный пожар относится к Е классу – пожару горючих веществ и материалов электроустановок, находящихся под напряжением. В качестве первичных средств пожаротушения можно выделить порошковые огнетушители для установок до 1000 В, углекислые огнетушители для установок до 10000 В [32].

Заключение

В ходе выполнения раздела по социальной ответственности были выявлены вредные и опасные производственные факторы. Фактические значения искусственного освещения и высоты рабочей поверхности соответствуют нормативным значениям.

Категория учебного помещения по электробезопасности, согласно ПУЭ, относится к первому классу – помещения без повышенной опасности [33].

Персонал по электробезопасности согласно Правил по охране труда при эксплуатации электроустановок должен обладать I группой допуска [34].

Категория тяжести труда по СанПиН 1.2.3685-21 «Гигиенические нормативы и требования к обеспечению безопасности и (или) безвредности для человека факторов среды обитания» относится к категории Iб – работы, производимые сидя, стоя или связанные с ходьбой и сопровождающиеся физическим напряжением [20].

Категория помещения по взрывопожарной и пожарной опасности согласно СП 12.13130.2009 «Определение категорий помещений, зданий и наружных установок по взрывопожарной и пожарной опасности» относится к группе Д, возможный класс пожара Е [35].

Объект, оказывающий незначительное негативное воздействие на окружающую среду, относится к IV классу опасности [36].

ЗАКЛЮЧЕНИЕ

В ходе выполнения выпускной квалификационной работы было проведено изучение используемых в работе технологий, а также подходов к разработке программного обеспечения и на основании изученного материала был разработан прототип системы для захвата, обработки и визуализации электрических параметров сварочного процесса.

Был проведен ряд испытаний, призванных подтвердить работоспособность разработанной программы, а также выявить возможные ошибки. В ходе проведения испытаний были сняты, а после проанализированы, показания осциллографа с процесса ручной дуговой сварки покрытым электродом. Результаты испытаний удовлетворяли требованиям к программе, а потому были признаны успешными.

Также были разработаны разделы «Финансовый менеджмент, ресурсоэффективность и ресурсосбережение» и «Социальная ответственность», в которых описываются экономическая состоятельность представленной программы и мероприятия по снижению вероятности возникновения опасных для жизни и здоровья происшествий при разработке или эксплуатации.

Подводя итог, можно отметить, что разработанный прототип может стать основой для создания более обширных и комплексных систем для более глубокого изучения и анализа сварочных процессов за счет большей гибкости и открытости пользователям, возможности использования собственноручно написанных скриптов, а также отсутствии необходимости в дополнительном оборудовании связанным с программой – будет достаточно осциллографа.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Что такое .NET и чем занимаются .NET-разработчики: сайт. – URL: https://training.epam.ua/News/Items/301?lang=ru (дата обращения: 10.03.2023). – Текст: электронный.
2. Прайс Марк C# 10 и .NET 6. Современная кросс-платформенная разработка. — СПб.: Питер, 2023. — 848 с.: ил. — (Серия «Для профессионалов»). ISBN 978-5-4461-2249-3
3. Троелсен, Э., Джепикс, Ф. Язык программирования C# 9 и платформа .NET 5: основные принципы и практики программирования, том 1, 10-е изд./Эндрю Троелсен, Филипп Джепикс; пер. с англ. Ю.Н. Артеменко. — Киев.: “Диалектика”, 2022.— 770 с.: ил. — Парал.тит. англ. ISBN 978-617-7987-81-8 (том 1)
4. Руководство по классическим приложениям (Windows Forms .NET): сайт. – URL: https://learn.microsoft.com/ru-ru/dotnet/desktop/winforms/overview/?view=netdesktop-7.0 (дата обращения: 10.03.2023). – Текст: электронный.
5. Симан Марк, Дерсен Стивен ван Внедрение зависимостей на платформе .NET. 2-е издание. — СПб.: Питер, 2021. — 608 с.: ил. — (Серия «Для профессионалов»). ISBN 978-5-4461-1166-4
6. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. Паттерны объектно-ориентированного проектирования. — СПб.: Питер, 2020. — С. 38.
7. Metanit.com. Сайт о программировании: сайт. – URL: https://metanit.com/ (дата обращения: 10.03.2023). – Текст: электронный.
8. Тюкачев Н. А. C#. Алгоритмы и структуры данных: учебное пособие для СПО / Н. А. Тюкачев, В. Г. Хлебостроев. — Санкт-Петербург: Лань, 2021. — 232 с.: ил. + CD. — Текст: непосредственный. ISBN 978-5-8114-6817-1

9. Альтхофф Кори Computer Science для программиста-самоучки. Все, что нужно знать о структурах данных и алгоритмах. — СПб.: Питер, 2023. — 240 с.: ил. — (Серия «Библиотека программиста»). ISBN 978-5-4461-2010-9
10. Студми. Учебные материалы для студентов: сайт. – URL: https://studme.org/ (дата обращения: 20.12.2021). – Текст: электронный.
11. Смит Дж. П. Entity Framework Core в действии / пер. с англ. Д. А. Беликова. – М.: ДМК Пресс, 2022. – 690 с.: ил. ISBN 978-5-93700-114-6
12. Российская Федерация. Законы. Трудовой кодекс Российской Федерации: Федеральный закон №197-ФЗ: [принят Государственной Думой 30 декабря 2001 года]. – Текст: электронный. – URL: https://www.consultant.ru/document/cons_doc_LAW_34683/ (дата обращения: 25.04.2023).
13. Российская Федерация. Законы. О специальной оценке условий труда: Федеральный закон №426-ФЗ: [принят Государственной Думой 28 декабря 2013 года]. – Текст: электронный. – URL: https://www.consultant.ru/document/cons_doc_LAW_156555/ (дата обращения: 25.04.2023).
14. ГОСТ 12.2.032-78. Система стандартов безопасности труда. Рабочее место при выполнении работ сидя. Общие эргономические требования: дата введения 1979-01-01. – URL: https://internet-law.ru/gosts/gost/31970/ (дата обращения: 25.04.2023). – Текст: электронный.
15. ГОСТ 22269-76. Система «Человек-машина». Рабочее место оператора. Взаимное расположение элементов рабочего места. Общие эргономические требования: дата введения 1978-01-01. –

URL: https://internet-law.ru/gosts/gost/33818/ (дата обращения: 25.04.2023). – Текст: электронный.

16. ГОСТ Р 50923-96. Дисплеи. Рабочее место оператора. Общие эргономические требования и требования к производственной среде. Методы измерения: дата введения 1997-07-01. – URL: https://internet-law.ru/gosts/gost/5265/ (дата обращения: 25.03.2023). – Текст: электронный.
17. Санитарно-эпидемиологические требования к условиям труда: (СП 2.2.3670-20): официальное издание: утверждены постановлением Главного государственного санитарного врача Российской Федерации от 2.12.2020: введены в действие 01.01.2021. – Москва, 2023. – 64 с.; (Санитарные правила). – ISBN 978-5-903070-26-8.
18. ГОСТ 12.0.003-2015. Система стандартов по безопасности труда. Опасные и вредные производственные факторы. Классификация: дата введения 2017-03-01. – URL: https://internet-law.ru/gosts/gost/62075/ (дата обращения: 28.04.2023). – Текст: электронный.
19. Естественное и искусственное освещение (СП 52.13330.2016): официальное издание: утвержден приказом Министерства строительства и жилищно-коммунального хозяйства Российской Федерации от 07.11.2016: введены в действие 08.05.2017. – Москва, 2023. – 165 с.: (Свод правил).
20. Гигиенические нормативы и требования к обеспечению безопасности и (или) безвредности для человека факторов среды обитания: (СанПиН 1.2.3685-21): официальное издание: утверждены постановлением Главного государственного санитарного врача Российской Федерации от 28.01.2021: введены в действие 01.03.2021. – Москва, 2023. – 736 с.: (Санитарные правила и нормы). – ISBN 978-5-908080-75-0.

21. Профилактика стрессового состояния работников при различных видах профессиональной деятельности (МР 2.2.9-2311-07): официальное издание: утверждены Главным государственным санитарным врачом Российской Федерации от 18.12.2007: введены в действие 18.03.2008. – Москва, 2023. – 52 с.: (Методические рекомендации). – ISBN 978-5-908080-23-3.
22. ГОСТ 31610.32-1-2015. Взрывоопасные среды. Часть 32-1. Электростатика. Опасные проявления. Руководство: дата введения 2017-12-01. – URL: https://internet-law.ru/gosts/gost/61974/ (дата обращения: 31.04.2023). – Текст: электронный.
23. ГОСТ 12.1.045-84. Система стандартов безопасности труда. Электростатические поля. Допустимые уровни на рабочих местах и требования к проведению контроля: дата введения 1985-06-30. – URL: https://internet-law.ru/gosts/gost/2729/ (дата обращения: 31.04.2023). – Текст: электронный.
24. ГОСТ 12.4.124-83. Система стандартов безопасности труда. Средства защиты от статического электричества. Общие технические требования: дата введения 1984-01-01. – URL: https://internet-law.ru/gosts/gost/21169/ (дата обращения: 31.04.2023). – Текст: электронный.
25. ГОСТ 12.1.009-2017. Система стандартов безопасности труда. Электробезопасность. Термины и определения: дата введения 2019-01-01. – URL: https://internet-law.ru/gosts/gost/70057/ (дата обращения 30.04.2023). – Текст: электронный.
26. ГОСТ 12.4.011-89. Система стандартов безопасности труда. Средства защиты работающих. Общие требования и классификация: дата введения 1990-06-30. – URL: https://internet-law.ru/gosts/gost/11167/ (дата обращения: 29.04.2023). – Текст: электронный.
27. ГОСТ 12.1.038-82. Система стандартов безопасности труда. Электробезопасность. Предельно-допустимые значения напряжений

прикосновения и токов: дата введения 1983-06-30. – URL: https://internet-law.ru/gosts/gost/21681/ (дата обращения: 30.04.2023). – Текст: электронный.

28. ГОСТ 12.1.007-76 ГОСТ 12.1.007-76 Система стандартов безопасности труда (ССБТ). Вредные вещества. Классификация и общие требования безопасности (с Изменениями N 1, 2) – URL: https://docs.cntd.ru/document/5200233?ysclid=lhhxutqsrq223141146&section=status– Текст: электронный.
29. Российская Федерация. Законы. Об охране атмосферного воздуха: Федеральный закон №96-ФЗ: [принят Государственной Думой 2 апреля 1999 года]. – Москва, 2023. – 40 с. – ISBN: 978-5-703080-24-5.
30. ГОСТ Р 53692-2009. Ресурсосбережение. Обращение с отходами. Этапы технологического цикла отходов: дата введения 2011-01-01. – URL: https://internet-law.ru/gosts/gost/50159/ (дата обращения: 02.05.2023). – Текст: электронный.
31. ГОСТ 12.1.004-91. Система стандартов безопасности труда. Пожарная безопасность. Общие требования: дата введения 1992-06-30. – URL: https://internet-law.ru/gosts/gost/3254/ (дата обращения: 02.05.2023). – Текст: электронный.
32. Российская Федерация. Закон. Технический регламент о требованиях пожарной безопасности: Федеральный закон №123-ФЗ: [принят Государственной Думой 22 июля 2008 года]. – Москва, 2022. – 53 с.
33. Правила устройства электроустановок (ПУЭ). Глава 1.1 общая часть (Издание седьмое) (Правила устройства электроустановок): официальное издание: утверждено Министерством энергетики Российской Федерации от 08.07.2022: введены в действия 01.01.2003. – Москва, 2023. – 584 с.: (Правила). – ISBN 978-5-903089-16-1.
34. Правила по охране труда при эксплуатации электроустановок: официальное издание: утверждено Министерством труда и социальной защиты Российской Федерации от 15.12.2020: введены в

действия 01.01.2021. – Москва, 2023. – 166 с.: (Правила). – ISBN 978-5-903000-38-8.

35. Определение категорий помещений, зданий и наружных установок по взрывопожарной и пожарной опасности (СП 12.13130.2009): официальное издание: утверждено приказом МЧС России от 25.03.2009: введены в действия 01.02.2011. – Москва, 2023. – 28 с.: (Свод правил).
36. Критерии отнесения объектов, оказывающих негативное воздействие на окружающую среду, к объектам I, II, III и IV категорий (Критерии): официальное издание: утверждено Правительством Российской Федерации от 31.12.2020: введены в действия 01.01.2021. - Москва, 2023. – 16 с.: (Критерии). – ISBN 978-5-902080-71-8.

Приложение А

(справочное)

Development of a technology for capturing, processing and visualizing the electrical parameters of the welding mode

Обучающийся

Группа	ФИО	Подпись	Дата
1ВМ11	Иванов Владимир Николаевич		

Руководитель ВКР

Должность	ФИО	Ученая степень, звание	Подпись	Дата
доцент ОЭИ	Гордынец Антон Сергеевич	к.т.н.		

Консультант-лингвист отделения иностранных языков ШБИП

Должность	ФИО	Ученая степень, звание	Подпись	Дата
Доцент ОИЯ	Щеголихина Юлия Викторовна	к.ф.н.		

ABSTRACT

Construction, automotive and aviation industries, and many others cannot do without metal welding. Welding is one of the most important stages of production at the moment. Depending on the production conditions, a variety of welding methods are used: from welding with a coated electrode or friction, to electron beam or explosion. The welding process has firmly established itself in production and with the development of equipment, development is required for the welding process itself.

Now, many companies, both domestic and foreign, producing welding equipment are engaged in improving the welding process. "New" welding methods are being developed, often only slightly different from each other. In order to make progress in this direction, it is necessary to begin to conduct a more detailed analysis of the welding process.

In this final qualification work, for the first step in the development of welding process data analysis tools, a developed software prototype is proposed, which, together with the recording equipment (oscilloscope), will allow you to see in more detail and analyze the incoming data on the welding process.

In connection with the above, the purpose of this work is: the choice of a tool for software prototyping, the analysis of the available literature on working with the selected tool, the development of a prototype with subsequent testing.

To achieve this goal, it is necessary to solve the following tasks: select a tool for prototyping the application, develop modules for reading, processing, storing and visualizing the data being captured, develop a module for connecting third-party data analysis tools, and test the prototype.

1 LITERATURE REVIEW

In order to start prototyping a project, you need to decide what tools will be used during development: a programming language, a framework and a library, patterns that make the task easier, etc.

This section will present the selected tools used in the performance of work, as well as provide introductory information on the algorithms.

1.1 Platform and programming language

When choosing a platform and programming language for the implementation of your project, you must consider its goals, performance requirements, scalability and security, as well as integration with other technologies. The .NET platform and the C# programming language are powerful tools for creating applications and have many advantages that can be used in various areas, from developing desktop applications to creating web services and mobile applications.

1.1.1 .Net platform

The .NET Framework is Microsoft's platform for software development, first released along with the C# programming language in 2002. Similar in many respects to Java, for a long time it was considered its direct competitor and alternative. The main difference from Sun's product is the focus on working primarily with Microsoft Windows operating systems. To date, Microsoft has stopped its development, concentrating on the new generation - .Net Core, but despite the long absence of updates, it is very popular: an impressive number of products have been created, and an extensive and loyal community still supports its projects developed under the .Net Framework.

The .Net Framework was replaced in 2016 by .Net Core (now renamed simply .Net), which is based on the idea of cross-platform, which allows you to create projects on it for various operating systems and equipment: from smart TVs to washing machines. Since then, Microsoft's platform has become more attractive to businesses, as it allows them to cover an even wider range of tasks.

Cross-platform modern .Net is possible due to the translation of application code written in one of the platform languages into Intermediate Language (IL) code and saves it in an assembly (DLL or EXE file).

Intermediate language (IL) code statements are similar to assembly language code, only executed by the CoreCLR virtual machine in .NET. At runtime, IL code is loaded by the CoreCLR from the assembly, dynamically (just-intime, JIT) compiled by the compiler into native CPU instructions, and then executed by the CPU on your computer.

The benefit of this three-step compilation process is that Microsoft can build the CLR not only for Windows, but also for Linux and macOS. The same IL code runs in any environment thanks to a second compilation process that generates code for a specific operating system and CPU instruction set. [1]

.Net Core provides developers with an easy-to-use platform for building Windows Forms, WPF or UWP desktop applications, web applications, services, libraries, and more. Also, the updated .Net Core provides the possibility of two-way interaction between it and the .Net Framework due to the .Net Standard system. .Net supports several programming languages (C#, F#, and VB.Net). The .Net base class library provides many prebuilt classes for building applications. The .NET Core libraries are not registered in the system registry. What's more, .NET Core allows multiple versions of the framework and application to coexist harmoniously on the same machine. The .NET Core command-line interface (CLI) is a cross-platform tool chain for developing and packaging .NET Core applications. In addition to the standard tools provided with the .NET Core SDK, additional tools can be installed. [2]

.Net Core has three key (and related) components that make all of the above possible - Core Runtime (formally CoreCLR and CoreFX), CTS, and CLS. From an app developer's perspective, you can think of .NET Core as a runtime and a rich base class library. The runtime layer contains a set of minimal implementations that are specific to specific platforms (Windows, iOS, Linux) and architectures (x86, x64, ARM), as well as all base types for .NET Core.

One of the pillars of the .Net platform is CTS - the Common Type System - its specification describes all the data types and programming constructs that the runtime supports. It describes how types and program constructs interact with each other, as well as their record in the form of metadata.

Be aware that not all of the functionality defined in the CTS specification can be supported by a single .NET Core language. However, there is a Common Language Specification (CLS) that describes a subset of types and programming constructs that must be supported by all .NET Core programming languages. If you create .NET Core types that use only CLS-compliant features, you can be sure that all .NET Core languages can use them. However, if you use a data type or programming construct that is not supported by the CLS, then every .NET Core programming language cannot be guaranteed to use your .NET Core code library.

The set of base class libraries (BCLs) provided by the .NET Core framework is available to all .NET Core programming languages and includes many primitives such as streams, file I/O, graphics rendering systems, and interfacing, as well as support for various services needed for most applications. The base class libraries contain types that can be used to create any type of application and components that interact with each other.

Even with the release of .NET 7.0, the number of base class libraries in the .NET Framework far outnumbers those of its kind in .NET Core. Given the 14-year advantage of the .NET Framework over .NET Core, the situation is understandable. This mismatch creates problems when trying to use .NET Framework code with .NET Core code. The solution (and requirement) for .NET Framework/.NET Core interoperability is the .NET Standard.

The .NET Standard is a specification that defines the availability of .NET APIs and base class libraries that must be present in every implementation. The standard has the following characteristics:

defines a unified set of BCL APIs for all .NET implementations to be built regardless of workload;

allows developers to produce portable libraries that are consumable across all .NET implementations using the same set of APIs;

reduces or even eliminates conditional compilation of common source code for .NET APIs, leaving it only for operating system APIs.

The table in the documentation from Microsoft (https://docs.microsoft.com/en-us/dotnet/standard/net-standard) lists the minimum implementation versions that support each .NET Standard. [2]

1.1.2 C# programming language

Although the syntax of the C# language is similar to that of the Java language, it cannot be called a Java clone, since both languages belong to the C-based programming language family. In addition, C# borrows constructs from the VB and C++ languages, as well as functional programming languages such as LISP and Haskell. However, it was C-based languages that had the greatest influence on C#. C# has the ability to use the concepts of class properties, optional parameters, operator overloading, creating structures, enumerations, and callback functions. In addition, C# supports lambda expressions and anonymous types, which are found in functional programming languages. LINQ technology makes C# unique in the world of programming. In addition, there is no need for direct pointer manipulation in the language, and there is automatic memory management through garbage collection. There are also formal syntax constructs for classes, interfaces, structures, enums, and delegates, as well as support for attribute-based programming.

C# 9 is already a powerful language that, when combined with .NET Core, allows you to build a wide range of applications of various kinds.

C# is a programming language that can only be used to create software that runs on the .NET Core runtime. This means that C# cannot be used to create COM servers or unmanaged C/C++ style applications. Code written in C# that targets the .NET Core runtime is called managed code. Such code is contained in a binary module called an assembly. If the code cannot be directly maintained by the platform's management environment, then it is called unmanaged code.

As mentioned earlier, the .NET Core framework is capable of running on a variety of operating system environments. So it's entirely possible to build a C# app on a Windows machine using Visual Studio and run it on iOS using the .NET Core runtime. Alternatively, a C# application can be built on a Linux machine using Visual Studio Code and run on a Windows machine. With Visual Studio for Mac on a Mac, you can develop .NET Core apps that run on Windows, macOS, or Linux.

A C# program can still access unmanaged code, but then it will tie you to a specific development and deployment target. [2]

1.2 Windows Forms GUI framework

Windows Forms is a user interface framework that enables you to efficiently create Windows desktop applications using the visual designer in Visual Studio. It is a set of managed libraries for performing common tasks such as reading and writing to the file system. Windows Forms applications can be graphically complex and easy to deploy and update. A form in Windows Forms is a visual surface that displays information to the user, and controls are designed to enter and display data. Windows Forms has many controls, as well as drag and drop and align functions to quickly create complex form layouts. The System.Drawing namespace contains classes for drawing shapes on a form. [3]

Windows Forms provides an easy way to display data from various sources such as databases, XML or JSON files, web services, and others. The DataGridView control allows you to display tabular data in a traditional format, where each piece of data occupies its own cell. In addition, the DataGridView allows you to customize the appearance of the cells and fix the rows and columns in place.

Windows Forms also provides an easy way to connect to data sources over the network. The BindingSource component represents a connection to a data source and contains methods for binding data to controls, navigating between records, editing records, and saving changes to the original source. The BindingNavigator control provides a simple interface for navigating between records.

Using the Data Sources window in Visual Studio, you can easily create data-bound controls. This window displays existing data sources in the project, such as databases, web services, and objects. You can create data-bound controls by dragging objects from this window onto the project forms. You can also link existing controls to data by dragging objects from the Data Sources window onto existing controls.

Windows Forms also provides the ability to store some application state information at run time, such as the last known form size and user preferences. This can be done using application settings, which provide an easy way to store this information on the client computer in an XML file. Application settings are defined using Visual Studio or the code editor and are automatically read back into memory at runtime.

1.3 Approaches and patterns applicable in development

Many tasks have already been solved many times, for some more effective approaches and behaviors have been developed over time.

In this section, attention will be paid to the programming approaches and patterns used in the implementation of the project, general information about them will be given, and later, in the section on the progress of work, what the use of these patterns has yielded.

1.3.1 Dependency Injection

Dependency Injection or dependency injection is a set of software design principles and techniques that allows you to develop loosely coupled code.

This technology is not so much an end in itself, but a means to an end. Ultimately, the goal of most programming techniques is to provide a working software product in the most efficient way possible. At the same time, one of the aspects is writing maintainable code.

Pretty soon you'll find that the existing code needs to be extended and maintained. In general, the efficiency of working with such code is determined by the degree of its maintainability. [4]

A great way to increase maintainability is to loosen the binding. This was known as early as 1994 when the Gang of Four were working on the book Design Patterns: "program according to the interface, not the implementation." [5]

Loose coupling makes code extensible, which in turn makes it maintainable. DI is nothing but a loose coupling technology.

The use of DI is not an end goal, but a means to achieve a certain result. DI technology allows for loose coupling, which in turn makes code maintainable. [4]

1.3.2 MVVM pattern

The MVVM (Model-View-ViewModel) architectural pattern allows you to separate the internal logic of the software from its interface. MVVM consists of three components: Model (Model), View Model (ViewModel) and View (View).

A model is an object representation of data and can contain logic related to that data. Often, a model implements the INotifyPropertyChanged or INotifyCollectionChanged interfaces to notify the system of model property changes. This makes it easier to bind to the view, but there is no direct interaction between the model and the view.

The view is the application's user interface through which the user interacts with the application.

Any intelligence needs to be embedded somewhere else in the application. Only code in a code-behind file can be directly related to manipulating the user interface. It should not be based on business rules or anything else that needs to be protected for future use. While this is not the main purpose of MWM, well-designed MVVM applications usually have very little code-behind.

The view does not process events with rare exceptions, but performs actions mainly through commands.

The view model serves two purposes:

1. The view model offers a single location for all the data needed by the view. This does not mean that the view model is responsible for getting the actual data; instead, it is simply a transport mechanism for moving data from the store to the view. There is usually a one-to-one relationship between views and view models, but there are architectural differences that can vary on a case-by-case basis.

2. The second purpose of the view model concerns its action as a controller for the view. The view model takes instructions from the user and passes them to the appropriate code to perform the appropriate action. Quite often, such code takes the form of special commands.

The advantage of using this pattern is less coupling between components and the division of responsibility between them. That is, the Model is responsible for the data, the View is responsible for the graphical interface, and the ViewModel is responsible for the application logic.

Figure 1 shows a diagram of how the components interact with each other.

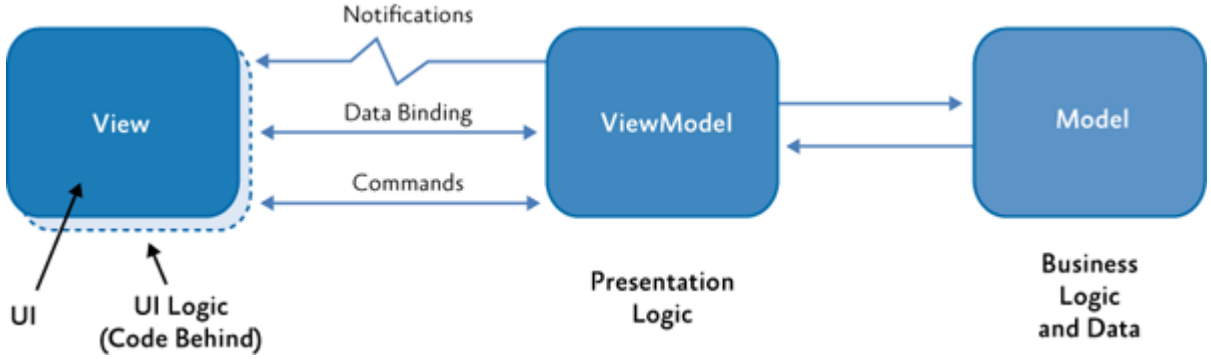


Figure 1 - Scheme of interaction of components

The result of applying the MVVM pattern is the functional division of the application into three components that are easier to develop and test, as well as further modify and maintain. [2]

1.4 Algorithms

An algorithm is a sequence of steps that solves a problem.

Despite the fact that algorithms are a fundamental concept in computer science, scientists have not yet come to a single definition. There are several competing formulations, but the one proposed by Donald Erwin Knuth is one of the most famous. It describes an algorithm as a defined, efficient, and final process that takes input and produces output based on the received input.

Certainty means that the steps are clear, concise and unambiguous.

Efficiency means that you can perform each operation exactly to solve the problem.

Finiteness implies that the algorithm stops after a certain number of steps.

A common addition to this list is correctness. The algorithm must always produce the same output for given inputs, and the output must be the correct answer to the problem the algorithm is solving.

Not all, but most of the algorithms meet these requirements, but some exceptions are important. Let's say when you create a random number generator, your goal is to generate randomness, so you can't use the output to figure out what the input was. In addition, many algorithms in data science do not adhere to correctness as strictly. For example, it may be sufficient for an algorithm to only assume a conclusion if its uncertainty is known in advance. However, in most cases, your algorithm should meet all the stated requirements.

1.4.1 Time complexity of the algorithm

One of the main criteria for evaluating an algorithm is its speed of solving a problem (time efficiency), but the complexity of its implementation is also important. Not always fast, but a complex algorithm is preferred, as a less complex algorithm can make debugging easier. The implementation of some algorithms may require additional memory to store intermediate data. [6]

For the analysis of algorithms, theoretical methods are used, the main of which is the method of asymptotic estimation of time efficiency. It consists in establishing the functional dependence of the number of actions performed by the algorithm on the amount of initial data in the limiting case of large amounts of this data. When

analyzing algorithms, their effectiveness is evaluated based on the assumption of the worst combination of input data values in terms of execution time. The time efficiency of algorithms can be expressed by polynomials of the second and higher degrees in the dimension of the problem. However, when passing to the limit of large n, all members of the polynomial are discarded, except for the highest one, which makes it possible to determine the growth rate for the algorithm - the law of change in the execution time on the dimension of the problem.

Since the important part of the algorithm is the part that grows faster as n increases, programmers use the "big O" notation instead of the equality T(n) to describe the efficiency of the algorithm. Big O notation is a mathematical notation that describes how, as n grows, the algorithm's time and memory requirements increase.

Programmers use Big O notation to create an order of magnitude function T(n). An order of magnitude is a type of object in a classification system in which each type is many times larger or smaller than the previous one. In an order of magnitude function, you use the portion of T(n) that dominates the equation and ignore the rest. The part of T(n) that dominates the equation is the order of magnitude of the algorithm. The following are generally accepted order-of-magnitude classifications for Big O notation, from best (most efficient) to worst (least efficient):

1. Constant time.
2. Logarithmic time.
3. Linear time.
4. Linear-logarithmic time.
5. Quadratic time.
6. Cubic time.
7. Exponential time.

Each order of magnitude describes the time complexity of the algorithm.

Time complexity is the maximum number of steps required for the algorithm to complete as n increases. [6]

Let's take a closer look at each order of magnitude.

Constant time

The most efficient order of magnitude is constant time complexity. The algorithm runs in constant time when it takes the same number of steps regardless of the size of the task. Big O notation for constant complexity is O(1).

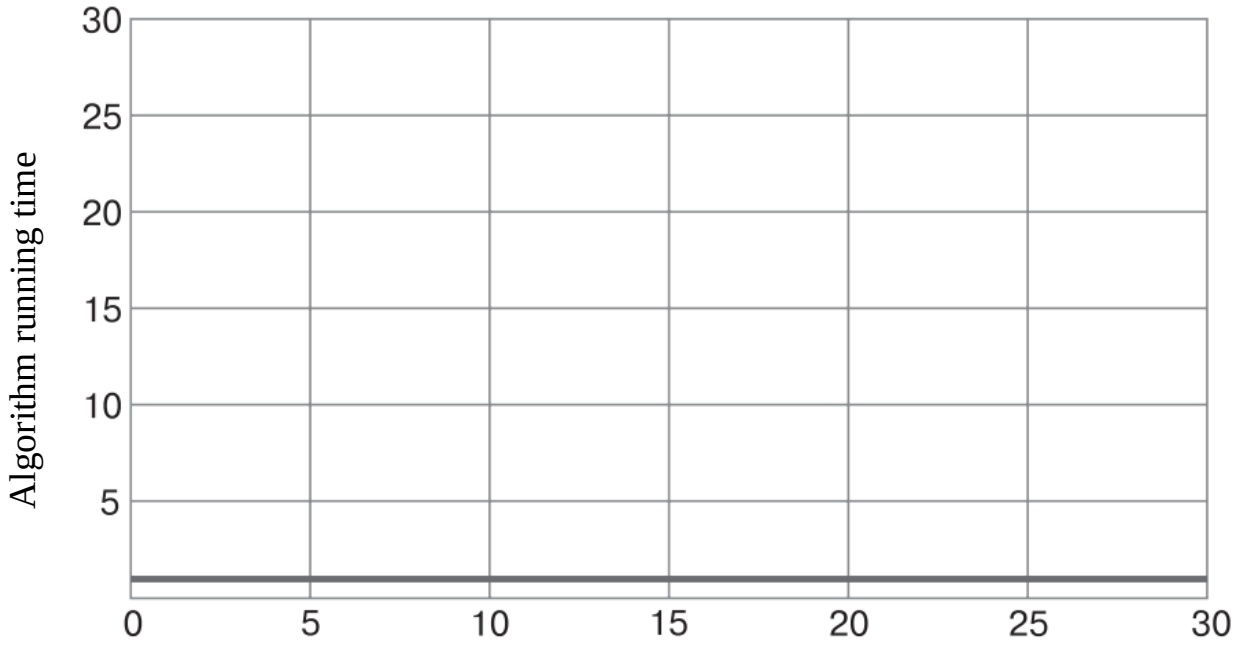


Figure 2 - Constant time complexity [6]

As you can see, the number of steps your algorithm needs to complete does not increase as the size of the task grows. Thus, this is the most efficient algorithm that can be written, because its time does not change with the increase in the data set.

logarithmic time

Logarithmic time is the second most efficient time complexity. The algorithm runs in logarithmic time, where the program execution time grows proportionally to the logarithm of the size of the input data. Similar time complexity can be seen in algorithms such as binary search, which may not consider many values in each loop. (If you don't understand something right now, don't worry—we'll discuss it in more detail later.) A logarithmic algorithm in Big O notation is expressed as O(log n).

Figure 3 shows how the logarithmic algorithm looks on the graph.

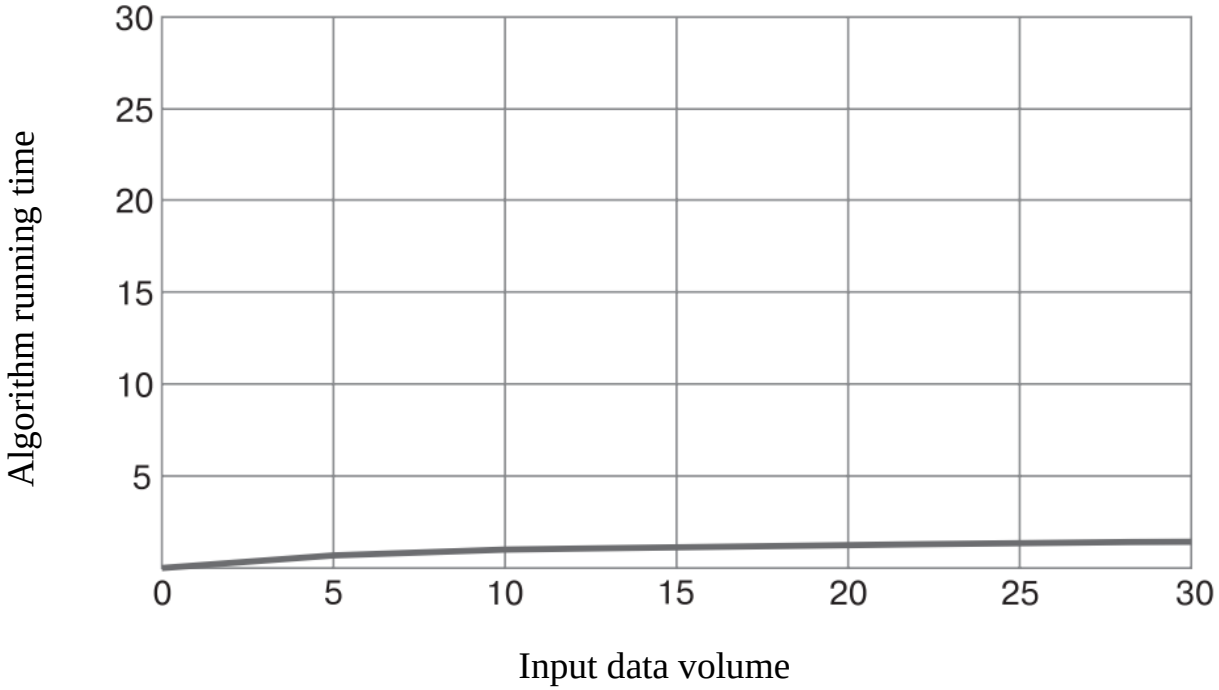


Figure 3 - Logarithmic time complexity [6]

In the logarithmic algorithm, the number of required steps grows rather slowly as the data set grows.

Linear time

The next most efficient type of algorithm is one that runs in linear time. Such an algorithm grows in proportion to the size of the problem. A linear algorithm in Big O notation is expressed as O(n).

In a linear algorithm, as n grows, the number of algorithm steps increases by the same amount as n (Figure 4).

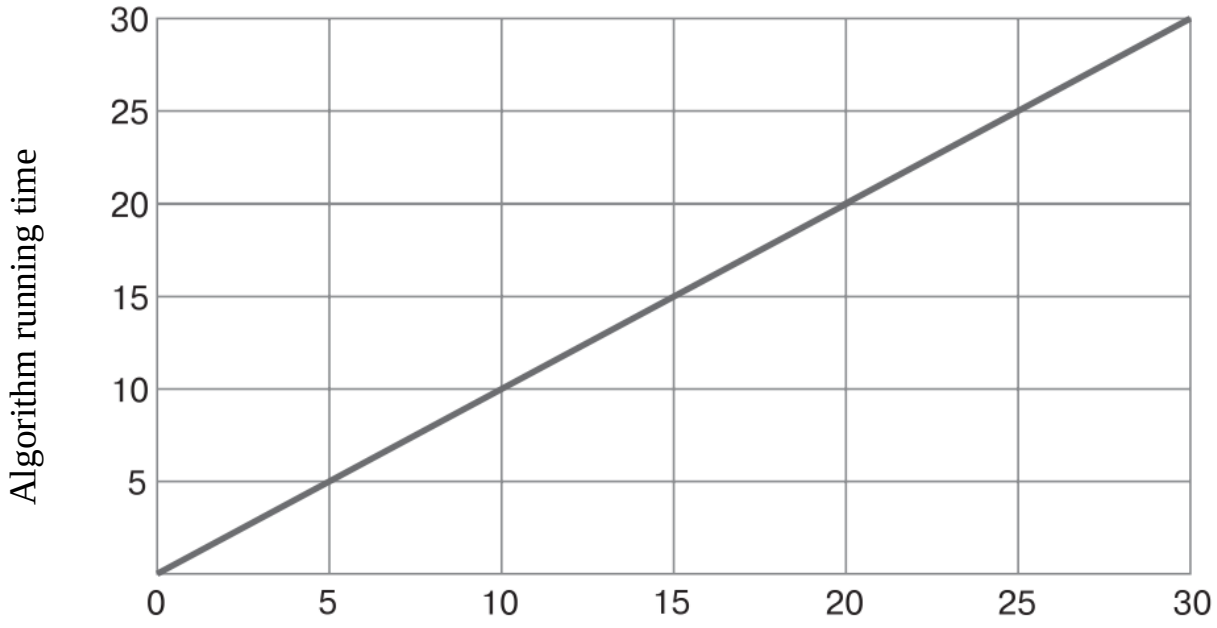


Figure 4 - Linear time complexity [6]

Linear-logarithmic time

An algorithm running in linear-logarithmic time grows as a combination (multiply) of logarithmic and linear time complexities. For example, a log-linear algorithm can evaluate the O(log n) operation n times. In Big O notation, the linear-logarithmic algorithm is expressed as O(n log n). Linear-logarithmic algorithms often divide the data set into smaller parts and process each of them separately. For example, most of the most efficient sorting algorithms, such as merge sort, are linear-logarithmic.

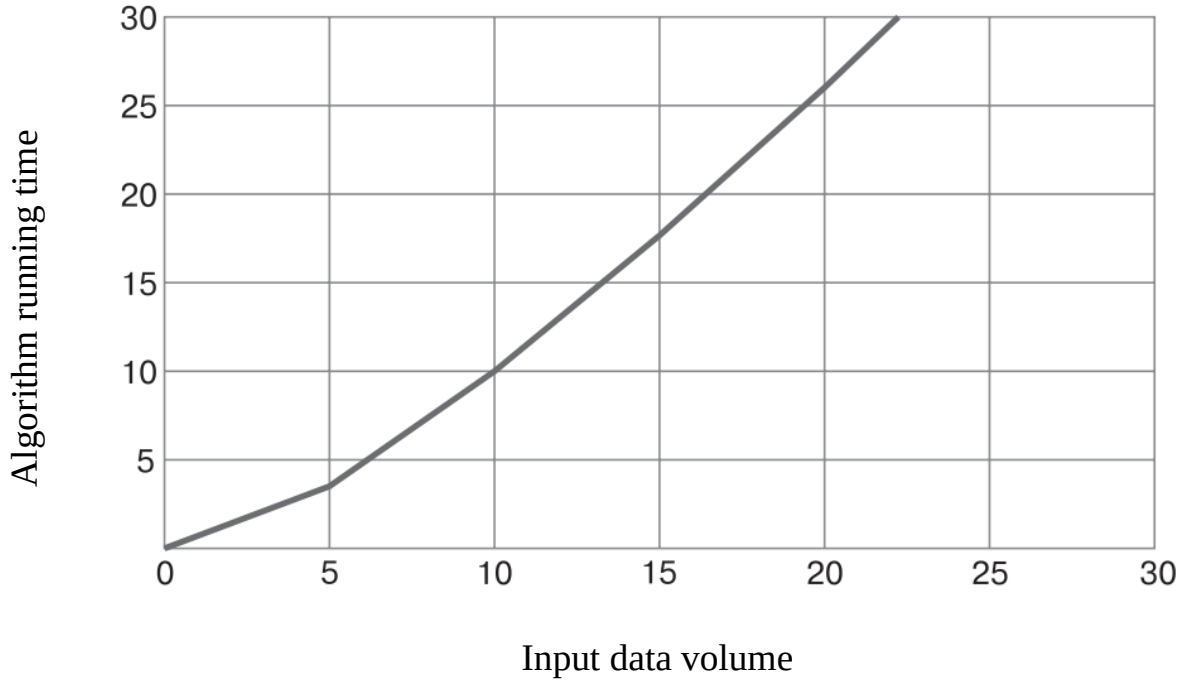


Figure 5 - Linear-logarithmic time complexity [6]

Quadratic time

Next in efficiency after linear-logarithmic time is quadratic time. The algorithm runs in quadratic time when its performance is directly proportional to the size of the problem squared. In Big O notation, the quadratic algorithm is expressed as O(n**2).

In the graph of the algorithm with quadratic time complexity, the number of steps increases dramatically as the size of the problem increases (Figure 6).

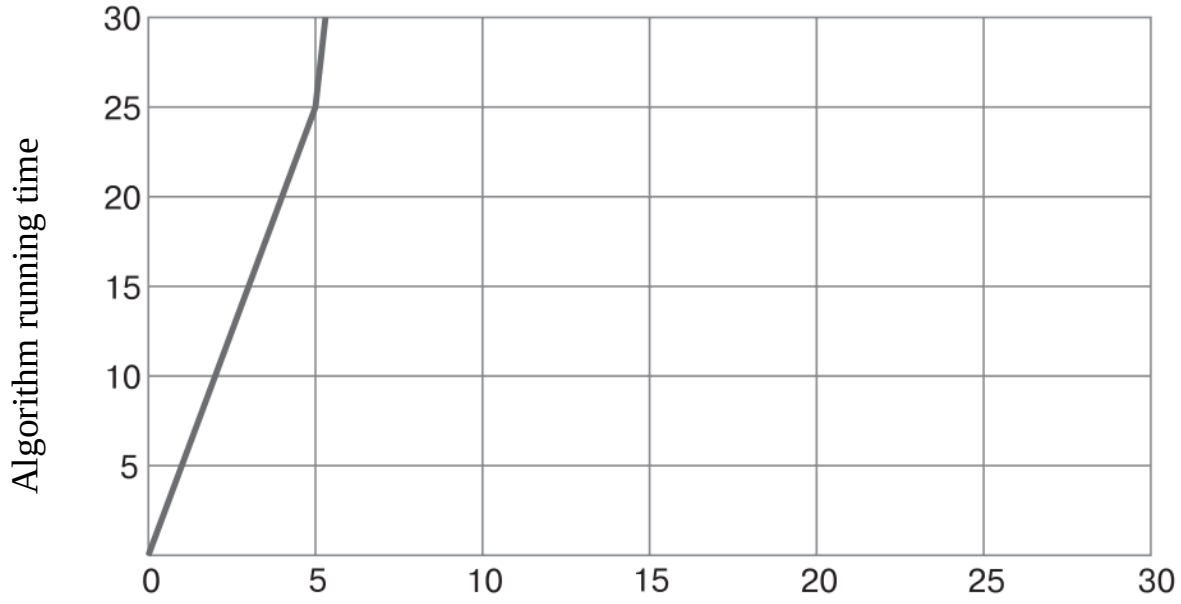


Figure 6 - Quadratic time complexity [6]

Typically, if your algorithm contains two nested loops running from 1 to n (or 0 to n - 1), the time complexity will be at least O(n**2). Most sorting algorithms, such as insertion sort or bubble sort, run in quadratic time.

Cubic time

Quadratic time complexity is followed by cubic time complexity. The algorithm runs in cubic time, where performance is directly proportional to the size of the task in the cube. In Big O notation, you express the cubic algorithm as O(n**3). The algorithm with cubic complexity is similar to the quadratic algorithm, only n is raised to the third power, and not to the second.

As in the algorithm with quadratic complexity, the most important part of this equation is n**3, which grows so fast that the rest of the equation, even the one that includes n**2, becomes irrelevant. Thus, in Big O notation, cubic complexity looks like this: O(n) = n**3

Two nested loops are indicative of quadratic time complexity, and three nested loops running from 0 to n are a sign of a cubic time algorithm.

Both quadratic and cubic time complexity are special cases of polynomial time complexity. An algorithm that runs in polynomial time is calculated as O(n**a), where a = 2 for quadratic time and a = 3 for cubic time. When creating an algorithm, as a rule, they try to avoid polynomial scaling, because as n increases, the execution time of the algorithm increases dramatically. [6]

Exponential time

An algorithm that runs in exponential time contains a constant that is scaled up to the size of the problem. In other words, an algorithm with exponential time takes steps raised to the power of n to complete. The Big O notation for exponential time is O(c**n), where c is a constant. The value of the constant does not matter. The only important thing is that n is in the exponent.

One example of this complexity associated with trying to guess a numeric password from n decimal places by checking all possible combinations is O(10**n).

Password Guessing is an example of a brute-force algorithm that checks all possible options. Brute force algorithms are generally inefficient and should only be chosen as a last resort.

Figure 7 shows a comparison of the efficiency of the considered types of algorithms.

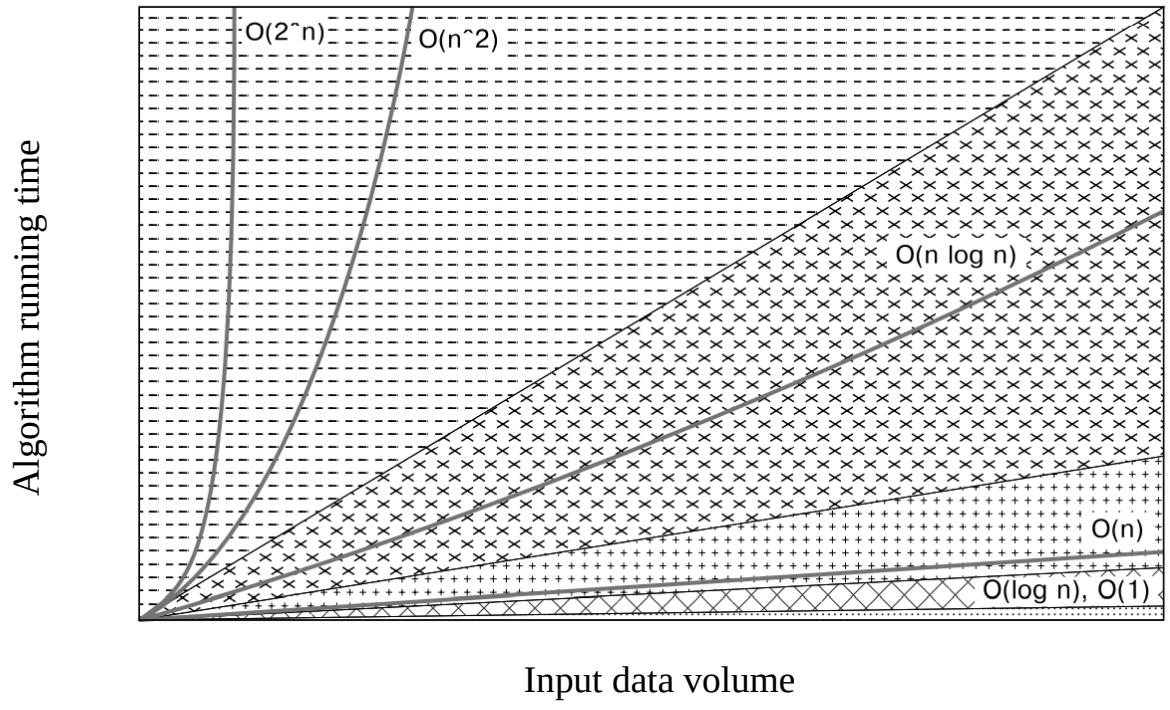


Figure 7 - Comparison of the effectiveness of different types of algorithms [6]

It follows that it is not necessary to rely on implementation details of an algorithm, such as the programming language or the computational capability of the hardware, to evaluate its time efficiency. Thanks to O-notation, it is possible to categorize algorithms based on their performance, regardless of other factors (input data, implementation details, etc.)

Knowing the rate of growth of the algorithm, it is possible to predict how many times the execution time of the algorithm will increase with a multiple increase in the dimension of the problem.

List of sources used

1. Price Mark C# 10 and .NET 6 - Modern Cross-Platform Development - Sixth Edition. Build apps, websites, and services with ASP.NET Core 6, Blazor, and EF Core 6 using Visual Studio 2022 and Visual Studio Code. – “Packt Publishing”, 2021. - 824 p. ISBN 978-1-801077-36-1
2. Jepix Philip, Troelsen Andrew. Pro C# 9 with .NET 5: Foundational Principles and Practices in Programming. – “Williams”, 2022. – 1392 p. ISBN 978-5-907458-67-3
3. Desktop Guide (Windows Forms .NET): website. – URL: https://learn.microsoft.com/en-us/dotnet/desktop/winforms/overview/?view=netdesktop-7.0 (date of access: 10.03.2023). – Text: electronic.
4. Mark Seemann. Dependency Injection in .NET. – “Manning”, 2011. – 584 p. ISBN 978-1-935182-50-4
5. Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. – “Addison-Wesley Professional”, 1994. – 416 p. ISBN 0-201-63361-2
6. Cory Althoff. The Self-Taught Programmer: The Definitive Guide to Programming Professionally. – “Self-Taught Media”, 2017. – 299 p. ISBN 978-1-119724-41-4