

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет
вычислительной техники

Кафедра
МОиПЭВМ

Направление подготовки 09.03.04 «Программная инженерия»

БАКАЛАВРСКАЯ РАБОТА

на тему

**«Программа для проверки внешних
накопителей»**

Студент

(подпись, дата)

Иванов Александр
Олегович

(ФИО полностью)

Руководитель

(подпись, дата)

Шашков Б.Д.

(фамилия, инициалы)

Нормоконтролёр

(подпись, дата)

Такташкин Д.В.

(фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от _____ № _____)

Заведующий кафедрой

(подпись)

Козлов А.Ю.

(фамилия, инициалы)

Работа защищена с отметкой _____ (протокол заседания ГЭК от _____ № _____)

Секретарь ГЭК

(подпись)

Попова Н.А.

(фамилия, инициалы)

Пенза, 2023

Федеральное государственное бюджетное образовательное учреждение
высшего образования
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет

Вычислительной техники

Кафедра

МО и ПЭВМ

«Утверждаю»
Заведующий кафедрой
_____ А.Ю. Козлов
«__» _____ 2023 г

**ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА**

1. Студент _____ Иванов Александр Олегович _____ гр. 19ВП2 факультета _____ ВТ
<Фамилия, имя, отчество полностью>

2. Тема работы «Программа для проверки внешних накопителей»

Тема утверждена приказом ПГУ № _____ от " _____ " _____ 20 _____

3. Руководитель работы Шашков Б.Д

4. Задание на работу (назначение разработки, исходные данные и т.п.)

Среда разработки: Windows 10, язык программирования C++17, платформа для создания графических приложений Qt 6.5.1, интерфейс программирования приложений Windows API.

Среда выполнения: персональные компьютеры с операционной системой Windows 10 или Windows 11.

Разработать программу для проверки внешних накопителей.

Основные функции приложения:

- проверка внешних накопителей на наличие плохих блоков;
- просмотр сведений о внешнем накопителе;
- проверка накопителя на плохие блоки без перезаписи и с перезаписью данных на накопителе;
- просмотр скорости чтения и записи данных на накопитель;
- возможность выбора записываемых данных на накопитель: случайных или шаблонных.

Технология разработки: ООП

Языки программирования: C++

5. Перечень подлежащих разработке вопросов

- 1) Анализ предметной области
- 2) Анализ функциональных требований
- 3) Планирование разработки и оценка бюджета
- 4) Проектирование приложения
- 5) Программная реализация
- 6) Контроль качества
- 7) Выполнение пояснительной записки

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области и требований к системе	10%	12.04.2023 – 16.04.2023	
2	Проектирование программных средств	20%	17.04.2023 – 26.04.2023	
3	Реализация программных средств	30%	27.04.2023 – 11.05.2023	
4	Тестирование программных средств	20%	12.05.2023 – 21.05.2023	
5	Оформление пояснительной записки	20%	22.05.2023 – 01.06.2023	

Дата выдачи задания «22» декабря 2022 г.

Руководитель бакалаврской работы _____
(подпись, дата)

Задание к исполнению принял студент _____
(подпись, дата)

Работу к защите допустить
Декан ФВТ _____ Л.Р. Фионова
(подпись, дата)

Реферат

Пояснительная записка содержит 104 листа, 21 рисунок, 12 таблиц, 20 использованных источников, 17 приложений.

АНАЛИЗ, ПРОЕКТИРОВАНИЕ, РАЗРАБОТКА, ТЕСТИРОВАНИЕ, ПРИЛОЖЕНИЕ, QT, WINDOWS, WINDOWS API, C++, C++17, ВНЕШНИЕ НАКОПИТЕЛИ, ПЛОХИЕ БЛОКИ.

Объект разработки – программа для проверки внешних накопителей на плохие блоки.

Цель работы – исследовать и проанализировать предметную область, выделить требования, спроектировать, реализовать и протестировать программный продукт.

Технология разработки: C++17, Qt 6, Windows API.

В результате выполнения работы было спроектировано и создано приложение для проверки внешних накопителей на плохие блоки.

					ПГУ 09.03.04 – 8ВР192.09 ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата				
Разраб.		Иванов А.О.			«Программа для проверки внешних накопителей» Пояснительная записка.	Лит.	Лист	Листов
Провер.		Шашков Б.Д.					4	104
						Группа 19ВП2		
Н. Контр.		Такташкин Д.В.						
Утверд.		Козлов А.Ю.						

Содержание

Содержание	5
Введение.....	8
1 Исследование предметной области	10
1.1 Введение в предметную область	10
1.2 Обзор программного обеспечения аналогичного назначения.....	10
1.2.1 Badblocks	10
1.2.2 H2testw.....	12
1.2.3 Bulldog	13
1.2.4 Flashnul	14
1.2.5 Сравнения аналогов	16
1.3 Анализ вариантов использования.....	17
1.4 Техническое задание	23
1.2.1 Функциональные требования к программе	23
1.2.2 Требования к дизайну приложения	23
1.2.3 Требования к дизайну приложения	24
1.2.4 Требования к программному обеспечению.....	24
1.2.5 Требования к дизайну приложения	24
1.5 Техническое задание	25
2 Проектирование приложения.....	27
2.1 Проектирование пользовательского интерфейса.....	27
2.2 Проектирование программных средств	28
3 Планирование разработки и оценка бюджета.....	33
3.1 Календарное и ресурсное планирование	33
3.2 Расчет бюджета	34

	6
4 Реализация приложения	39
4.1 Кодирование приложения	39
4.2 Кодирование алгоритмов.....	42
5 Анализ качества программного продукта	45
5.1 Тестирование программного продукта	45
5.1.1 Метод классов эквивалентности.....	45
5.1.2 Метод граничных условий	47
5.1.3 Функциональное тестирование.....	48
5.2 Метрики кода.....	48
5.2.1 Цикломатическая сложность	50
5.2.2 Метрики Холстеда.....	51
Заключение	54
Список использованных источников	56
Приложение А. Листинг кода файла windowsdeviceprovider.h	58
Приложение Б. Листинг кода файла windowsdeviceprovider.cpp	60
Приложение В. Листинг кода файла windowsdevice.h	62
Приложение Г. Листинг кода файла windowsdevice.cpp.....	64
Приложение Д. Листинг кода файла diskdevices.h	67
Приложение Е. Листинг кода файла diskdevices.cpp.....	69
Приложение Ж. Листинг кода файла handle.h	73
Приложение И. Листинг кода файла handle.cpp	75
Приложение К. Листинг кода файла volumesearchhandle.h	81
Приложение Л. Листинг кода файла volumesearchhandle.cpp	83
Приложение М. Листинг кода файла testworker.h	85
Приложение Н. Листинг кода файла testworker.cpp	88

Приложение П. Листинг кода файла testservice.h.....	93
Приложение Р. Листинг кода файла testservice.cpp.....	95
Приложение С. Результаты тестирования	98
Приложение Т. Диаграмма ганта.....	101
Приложение У. Граф потока управления	103

Введение

С постоянным развитием технологий хранения данных и увеличением объемов информации, внешние накопители стали неотъемлемой частью нашей повседневной жизни. Они предоставляют нам возможность переносить, сохранять и обмениваться данными на различных устройствах с удобством и надежностью. Однако, как и любая технология, они не застрахованы от возможных сбоев, ошибок или повреждений.

Когда на устройстве обнаруживается плохой блок, это означает, что любые данные, хранящиеся в этом блоке, могут быть потеряны или повреждены. Если на устройстве хранятся важные данные, такие как документы, фотографии или видео, потеря этих данных может быть катастрофической.

Чтобы избежать потери данных, рекомендуется прекратить использование устройства, если на нем обнаружатся плохие блоки, и принять меры по восстановлению данных [1].

Существуют специальные программы, которые могут помочь восстановить данные с устройства хранения, даже если на нем есть плохие блоки. Однако, чем дольше вы будете использовать устройство с плохими блоками, тем больше вероятность потери данных. Поэтому, если на устройстве были обнаружены плохие блоки, не следует откладывать восстановление данных на потом.

В связи с этим возникает необходимость в эффективных инструментах для проверки и обеспечения надежности внешних накопителей. Дипломная работа посвящена разработке программы проверки внешних накопителей, которая позволит проводить проверку накопителей путем записи на накопитель тестовых данных, их считывания и сравнения полученных данных с записанными.

Целью данной дипломной работы является создание эффективной и надежной программы проверки внешних накопителей, которая поможет пользователям и специалистам получить информацию о надежности их накопителей информации.

Для достижения поставленной цели в рамках дипломной работы были сформулированы следующие задачи:

- изучение литературы по проблеме наличия плохих блоков на внешних накопителях;
- анализ и обзор существующих методов и подходов в области проверки внешних накопителей;
- исследование предметной области;
- анализ и сравнение существующего программного обеспечения для проверки внешних накопителей;
- определение структуры и содержания программы проверки внешних накопителей для реализации функциональных требований;
- проектирование пользовательского интерфейса, программных средств и структур данных разрабатываемой программы проверки внешних накопителей;
- планирование и расчет бюджета разработки программы проверки внешних накопителей;
- реализация программы проверки внешних накопителей;
- проведение тестирования разработанной программы проверки внешних накопителей для проверки ее функциональности и надежности.

Выполнение данных задач позволит достичь целей дипломной работы и создать качественную программу проверки внешних накопителей.

1 Исследование предметной области

1.1 Введение в предметную область

Запоминающее устройство – это компьютерное аппаратное обеспечение, используемое для хранения, перемещения или извлечения файлов и объектов данных [2]. Оно может быть внутренним или внешним и принимать различные формы, такие как жесткие диски, оптические приводы, USB-накопители и другие.

«Плохой блок» – это участок на запоминающем устройстве, который не может быть использован для хранения данных из-за физических повреждений или ошибок в работе устройства. Такое может произойти вследствие падения устройства, электростатического разряда или других причин [3].

Сектор (блок) – это минимальная единица адресации на жестком диске, которая может хранить определенную часть информации [4].

Повреждение данных – это нарушение целостности информации на носителе, которое может быть вызвано не только плохими блоками, но и другими факторами, например, ошибками чтения/записи или воздействием вирусов.

Данные на диске – это информация, которая хранится на жестком диске компьютера или другого устройства. Это могут быть файлы, фотографии, видео и другие типы данных.

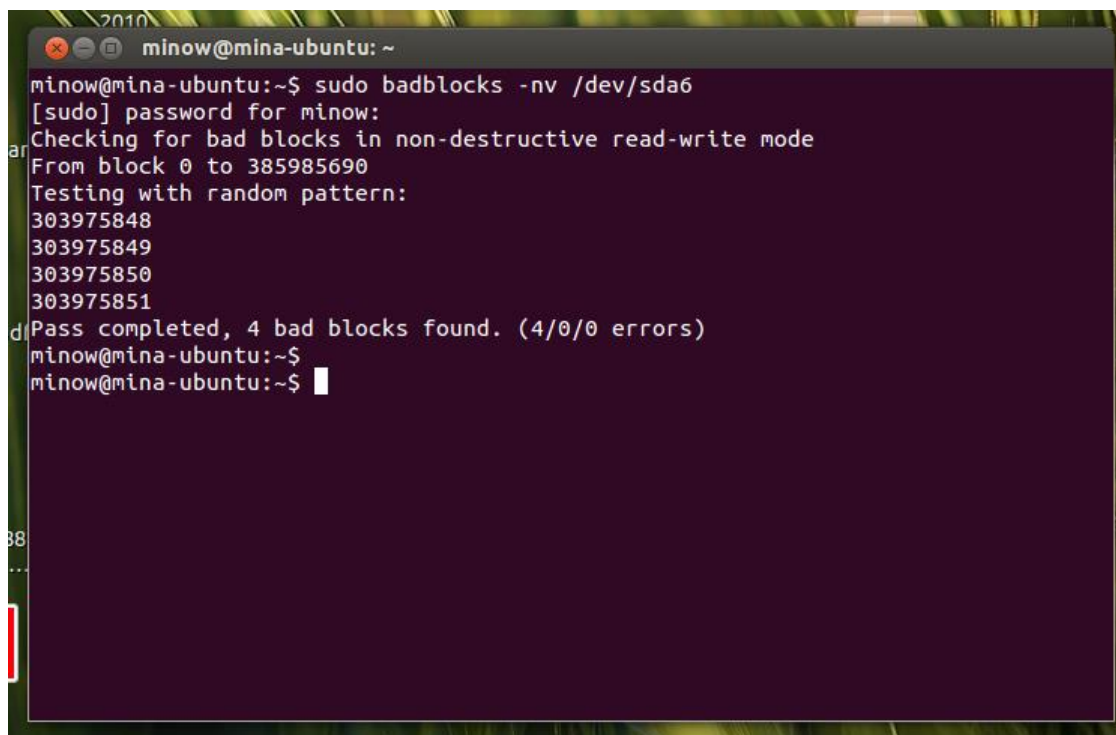
1.2 Обзор программного обеспечения аналогичного назначения

В современном мире существует огромное количество приложений, которые позволяют проверять внешние накопители на наличие неисправных блоков. Для того, чтобы провести сравнение программных продуктов, рассмотрим особенности основных представителей данного направления.

5.1.1 Badblocks

Badblocks – это консольное приложение (рисунок 1) с открытым исходным кодом для проверки блочных устройств в операционной системе Linux, которое

обеспечивает возможность проверки наличия сбойных блоков. Режимы работы приложения включают только чтение и чтение/запись, обеспечивая возможность записи определенных образцов в каждый блок на диске, а затем сравнения содержимого блока с записанным образцом при проверке наличия сбойных блоков в режиме чтения/записи. При проверке в режиме только для чтения приложение только читает содержимое блоков, не записывая никаких образцов.



```
minow@mina-ubuntu: ~  
minow@mina-ubuntu:~$ sudo badblocks -nv /dev/sda6  
[sudo] password for minow:  
Checking for bad blocks in non-destructive read-write mode  
From block 0 to 385985690  
Testing with random pattern:  
303975848  
303975849  
303975850  
303975851  
Pass completed, 4 bad blocks found. (4/0/0 errors)  
minow@mina-ubuntu:~$  
minow@mina-ubuntu:~$
```

Рисунок 1 – Результат выполнения программы badblocks

Badblocks поддерживает использование различных образцов при проверке наличия сбойных блоков, включая числовые значения или случайные биты. При проверке в режиме чтения/записи можно указать несколько образцов, каждый из которых будет использоваться для проверки блоков на диске.

Приложение имеет режим безопасной проверки, позволяющий проверить наличие сбойных блоков на диске без разрушения данных. В этом режиме приложение использует только операции чтения, что позволяет избежать потери данных на диске.

Badblocks может выводить информацию о процессе проверки на экран, что позволяет отслеживать процесс проверки и узнавать о наличии сбойных блоков

на диске. Приложение имеет режим реальной записи, который записывает определенные образцы в каждый блок на диске, но этот режим может быть опасен для данных на диске и должен использоваться с осторожностью.

Приложение поддерживает проверку нескольких устройств одновременно, что может быть полезно, если необходимо проверить несколько жестких дисков или других блочных устройств. Кроме того, Vadblocks может выводить подробную информацию о происходящем на экран, что может быть полезно для отладки проблем и выяснения подробностей об ошибках на накопителе [5].

5.1.2 H2testw

H2testw – это бесплатное приложение (рисунок 2) для тестирования накопителей на ошибки и проверки реального объема памяти. С помощью H2testw можно узнать, соответствует ли реальный объем памяти на накопителе заявленному производителем или продавцом. Некоторые производители или продавцы могут завышать объем памяти накопителя, что может привести к тому, что некоторые файлы могут быть случайно удалены или перезаписаны. H2testw может обнаружить такие проблемы и сообщить о них пользователю.

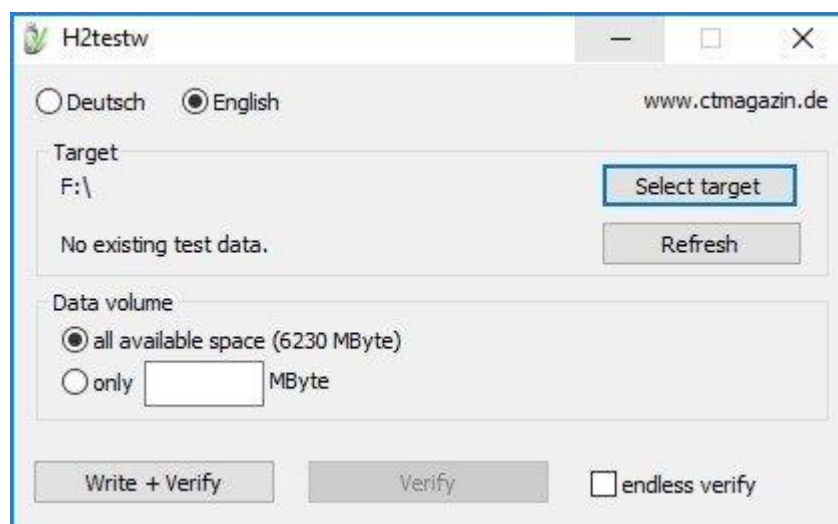


Рисунок 2 – Интерфейс приложения H2testw

H2testw генерирует случайные данные, записывает их в файлы определенного размера и сохраняет их на накопитель, пока накопитель не будет заполнен. Затем H2testw считывает файлы с накопителя и проверяет,

соответствуют ли они тем файлам, которые были записаны ранее. H2testw может использоваться для проверки любого типа накопителя, включая флеш-накопители, жесткие диски и карты памяти.

H2testw имеет простой и понятный интерфейс и не требует установки на компьютер. Приложение запускается в операционных системах Windows, начиная с Windows XP и выше, и может работать без обновлений. Однако H2testw имеет некоторые ограничения, например, он может перезаписывать файлы на проверяемом накопителе и не может использоваться для проверки накопителей, которые уже содержат данные.

В целом, H2testw – это простой и надежный способ проверки накопителей на ошибки и соответствие заявленному объему памяти. Если необходимо проверить, действительно ли накопитель может сохранить все данные, которые вы хотите записать на него, то H2testw подойдет для выполнения такой задачи [6].

5.1.3 Bulldog

Программа Bulldog (рисунок 3) представляет собой набор инструментов для проверки флеш-накопителей, SSD и HDD на наличие дефектов, искажений данных, скорости чтения и записи.

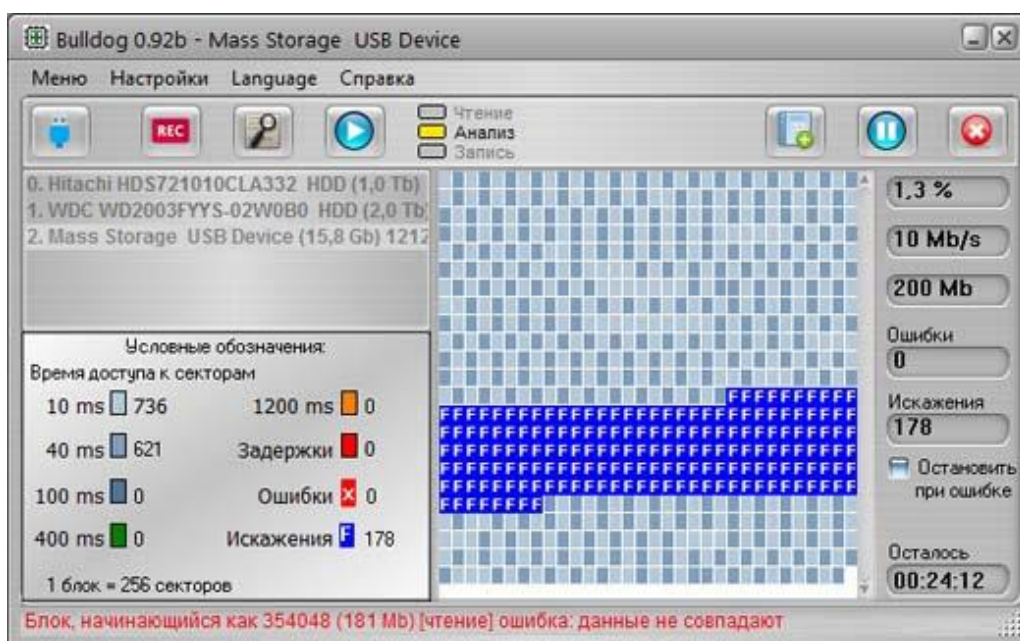


Рисунок 3 – Интерфейс приложения Bulldog

Основные возможности и особенности приложения Bulldog:

- проверка на подлинность: программа может обнаружить поддельные карты памяти, флеш-накопители и другие носители данных;
- проверка на наличие дефектов: Bulldog позволяет обнаружить физические дефекты, такие как повреждения секторов или проблемы с чтением и записью;
- измерение скорости чтения и записи: приложение может измерить скорость чтения и записи флеш-накопителя, чтобы убедиться, что он соответствует заявленным характеристикам;
- отчеты: после тестирования, приложение создает отчет, который позволяет увидеть результаты и анализ проверки;
- простота использования: для использования приложения не требуется специальных знаний, достаточно следовать инструкции;
- форматирование не требуется: программа работает напрямую с секторами, поэтому форматирование носителя перед работой не нужно;
- определение проблем с носителями: Bulldog поможет обнаружить такие проблемы, как потеря данных, мусорные символы в именах папок и файлов, разрушение файловой системы флешки и невозможность отформатировать накопитель;
- уничтожение информации: приложение также может стереть информацию с носителя без возможности ее восстановления.

Основная идея Bulldog заключается в том, чтобы помочь пользователям проверять носители данных на наличие проблем и убедиться, что они работают соответствующим образом. Она может быть полезна для пользователей, которые хранят важные данные на флеш-накопителях, SSD и HDD, а также для тех, кто хочет проверить свои носители данных на целостность [7].

5.1.4 Flashnul

Flashnul является консольной программой (рисунок 4) с открытым исходным кодом для проверки работоспособности и обслуживания различных

типов Flash-памяти, включая USB-Flash, IDE-Flash, SecureDigital, MMC, MemoryStick, SmartMedia, XD, CompactFlash и другие.

```
C:\Program Files\Far>flashnul \3 -TP

Disk PhysicalDrive3 <UNC name: \\.\PhysicalDrive3>
-----
DeviceIoCtrl & IOCTL_DISK_GET_DRIVE_GEOMETRY reports:
Cylinders          = 124
Tracks per cylinder = 255
Sectors per track   = 63
Bytes per sector     = 512
calculated disk size = 1019934720 <972Mb>
delta to near power of 2 = 53807104 <51Mb>
-----
data damaged:
  offset: 0x2e780000 <779616256>, block size: 65536
  error bytes: 64873
First 16 changed bytes (of 64873):
Offset (hex)  Write      Read      XOR
0x002e780000: 00000000 <0x00> : 01101101 <0xB6> : 01101101
0x002e780001: 11111111 <0xFF> : 11101111 <0xF7> : 00010000
0x002e780002: 00000000 <0x00> : 11111111 <0xFF> : 11111111
0x002e780003: 11111111 <0xFF> : 11111101 <0xBF> : 00000010
0x002e780004: 00000000 <0x00> : 11111101 <0xBF> : 11111101
0x002e780005: 11111111 <0xFF> : 11111110 <0x7F> : 00000001
0x002e780006: 00000000 <0x00> : 11111111 <0xFF> : 11111111
0x002e780008: 00000000 <0x00> : 11111110 <0x7F> : 11111110
0x002e78000a: 00000000 <0x00> : 11111111 <0xFF> : 11111111
0x002e78000b: 11111111 <0xFF> : 11111110 <0x7F> : 00000001
0x002e78000c: 00000000 <0x00> : 11111111 <0xFF> : 11111111
0x002e78000d: 11111111 <0xFF> : 11111101 <0xBF> : 00000010
0x002e78000e: 00000000 <0x00> : 11111111 <0xFF> : 11111111
0x002e78000f: 11111111 <0xFF> : 11111101 <0xBF> : 00000010
0x002e780010: 00000000 <0x00> : 11111111 <0xFF> : 11111111
0x002e780011: 11111111 <0xFF> : 11111110 <0x7F> : 00000001
Press ENTER to exit.
```

Рисунок 4 – Результат выполнения программы flashnul

Утилита Flashnul позволяет выполнить следующие действия:

- тест чтения: при выполнении данной операции программа проверяет доступность каждого сектора на носителе и выводит информацию о количестве ошибок чтения, если таковые имеются;

- тест записи: при выполнении данной операции программа записывает определенный шаблон данных на каждый сектор носителя и затем проверяет записанные данные, выводя информацию о количестве ошибок записи, если таковые имеются;

- тест сохранности записанной информации: данная операция позволяет проверить, сохраняется ли информация на носителе после записи и чтения – при выполнении данной операции программа записывает определенный шаблон данных на каждый сектор носителя, затем читает эти данные и сравнивает их с исходными, если данные сохранены корректно, программа выводит соответствующее сообщение;

– неdestructивная проверка чтения/записи: данная операция позволяет проверить доступность на запись каждого сектора устройства без потери записанных данных – при выполнении данной операции программа сначала считывает данные из каждого сектора устройства, затем записывает в каждый сектор определенный шаблон данных и проверяет доступность на запись;

– сохранение образа содержимого устройства: данная операция позволяет сохранить посекторный образ содержимого устройства в файл – при выполнении данной операции программа записывает содержимое каждого сектора устройства в отдельный файл, который затем можно использовать для восстановления данных;

– загрузка образа в устройство: данная операция позволяет записать посекторный образ содержимого устройства из файла обратно в устройство.

Данные операции могут быть полезны при диагностике и обслуживании Flash-памяти, а также при восстановлении данных [8].

5.1.5 Сравнения аналогов

После приведения нескольких аналогов с аналогичными или частично схожими функциями, необходимо сделать выводы. Для этого будет создана таблица (таблица 1), которая будет содержать основные критерии: наличие графического интерфейса пользователя, возможность destructive проверки, возможность неdestructивной проверки, возможность выбора образца для записи, измерение скорости чтения и записи, открытость исходного кода.

Таблица 1 – Сравнение аналогов

Приложение для обмена текстовыми сообщениями	Badblocks	H2testw	Bulldog	Flashnul	Разрабатываемое приложение
Графический интерфейс пользователя	-	+	+	-	+
Destructивная проверка	+	+	+	+	+
Неdestructивная проверка	+	-	+	+	+

Продолжение таблицы 1

Приложение для обмена текстовыми сообщениями	Badblocks	H2testw	Bulldog	Flashnul	Разрабатываемое приложение
Недеструктивная проверка	+	-	+	+	+
Возможность выбора образца для записи	+	-	-	+	+
Измерение скорости чтения и записи	+	+	+	+	+
Открытый исходный код	+	-	-	+	+

Было принято решение разработать собственное приложение, выполняющее функции, которые выполняются системами аналогичного назначения. Приложение будет распространяться под открытой лицензией и будет доступно в открытом репозитории в интернете.

1.3 Анализ вариантов использования

После проведения сравнения аналогов стало возможным выявить основные требования к разрабатываемому в рамках дипломной работы приложению:

- приложение должно иметь графический интерфейс пользователя, который должен обеспечивать возможность использовать всю функциональность приложения;
- приложение должно обеспечивать возможность недеструктивной проверки запоминающего устройства путем проверки доступности блока данных;
- приложение должно обеспечивать возможность деструктивной проверки запоминающего устройства путем записи определенных данных на диск, дальнейшего считывания данных и их сверки с ранее записанными данными;
- приложение должно обеспечивать возможность выбора данных, которые будут записаны на диск: данные могут быть полностью случайными, либо быть

образцом определенного размера, которым будет заполнен весь объем запоминающего устройства;

- приложение должно отображать скорость записи тестовых данных на диск в случае, если проводится запись тестовых данных;

- приложение должно отображать скорость чтения тестовых данных с диска в момент считывания данных с диска.

В качестве дополнительной функциональности, которой нет в программном обеспечении аналогичного назначения было принято решение дать пользователю возможность выбора объема записываемых на запоминающее устройство тестовых данных. Такая функциональность позволит пользователю не ожидать длительной записи полного объема диска – это позволит пользователю получить приблизительную оценку надежности запоминающего устройства, достаточную в некоторых сценариях использования запоминающего устройства. Тестовые данные, записанные в таком случае, будут равномерно распределены по всему объему запоминающего устройства.

После выявления основных требований к разрабатываемому приложению возможно выявить все возможные варианты использования приложения (рисунок 5):

- выбор тестируемого накопителя: для того, чтобы провести тестирование накопителя необходимо выбрать накопитель из списка доступных для чтения или записи накопителей на устройстве, накопители, которые не предполагают возможность записи на них данных будут недоступны для деструктивной проверки;

- просмотр сведений о тестируемом накопителе: пользователь сможет просмотреть сведения о тестируемом накопителе – производителя, модель, объем, текущую скорость записи, текущую скорость чтения;

- выбор типа проверки: пользователю будет доступна возможность выбора типа проверки тестируемого накопителя: деструктивная и недеструктивная проверка, недеструктивная проверка будет полезна для запоминающих устройств, которые имеют возможности самодиагностики и сообщают

операционной системе персонального компьютера о наличии плохих блоков на них;

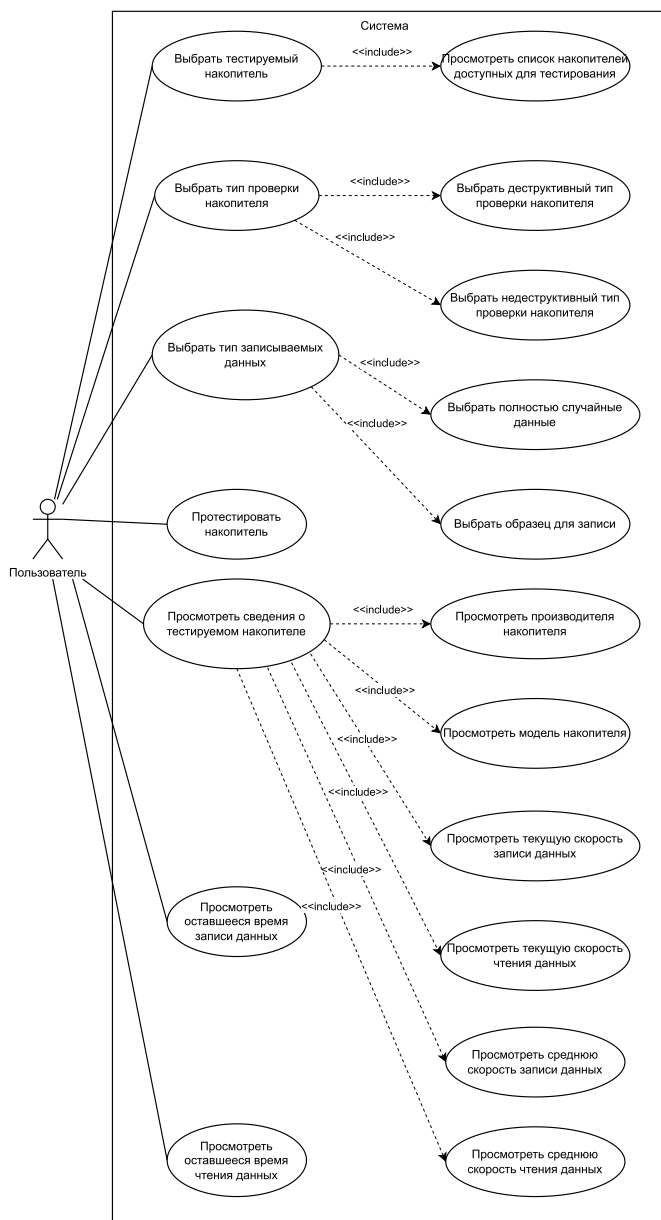


Рисунок 5 – Диаграмма вариантов использования

– выбор типа записываемых данных на запоминающее устройство: пользователю будет доступна возможность выбора записываемых данных – полностью случайные данные, либо периодически повторяющийся выбранный образец данных;

– просмотр скорости записи данных: пользователю будет доступна возможность просмотра скорости записи тестовых данных на диск;

– оценка оставшегося времени записи данных: имея данные о скорости записи тестовых данных будет возможно выявить приблизительное время, оставшееся до заполнения диска тестовыми данными и отобразить вычисленное время пользователю;

– просмотр скорости чтения данных: пользователю будет доступна возможность просмотра скорости чтения тестовых данных с диска;

– оценка оставшегося времени чтения данных: имея данные о скорости чтения тестовых данных будет возможно выявить приблизительное время, оставшееся до полного прочтения тестовых данных с тестируемого накопителя и отобразить вычисленное время пользователю.

После анализа диаграммы вариантов использования составим диаграмму деятельности (рисунок 6).

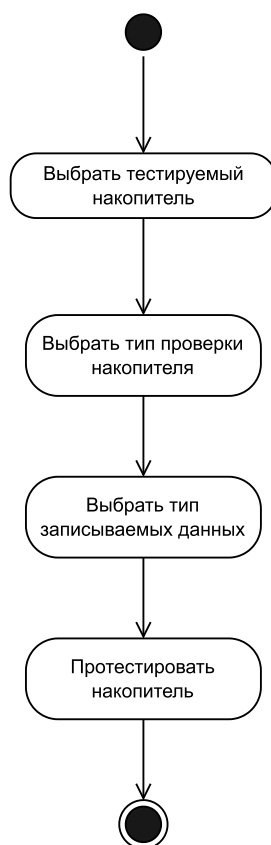


Рисунок 6 – Диаграмма деятельности

Основываясь на диаграмме вариантов использования и диаграмме деятельности, можно сформировать следующие диаграммы последовательности действий:

– для прецедента «Выбор тестируемого накопителя» (рисунок 7);

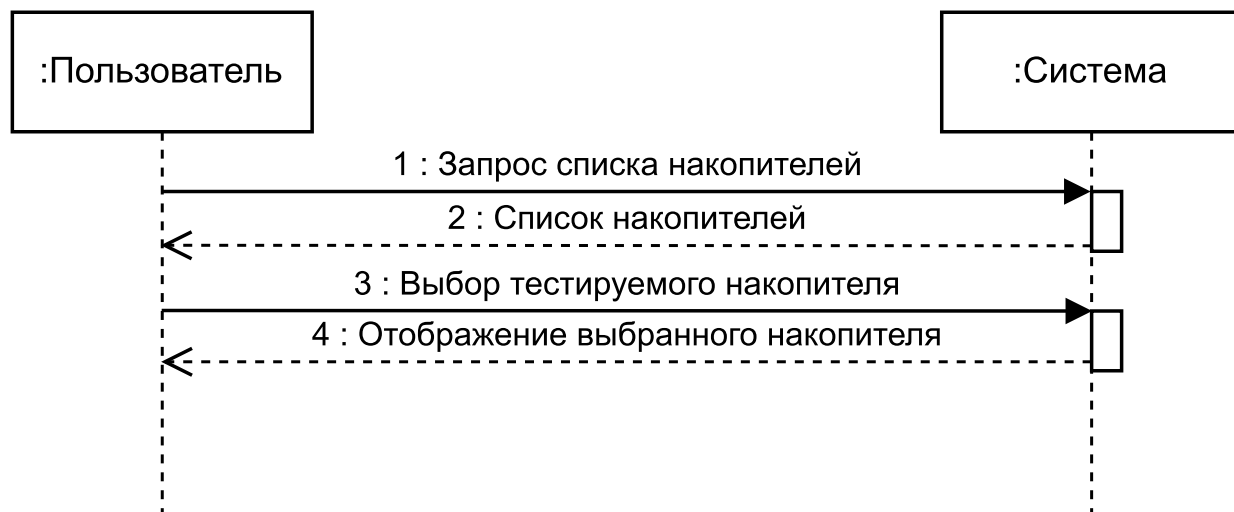


Рисунок 7 – Диаграмма последовательности действий для прецедента «Выбор тестируемого накопителя»

– для прецедента «Выбор типа проверки накопителя» (рисунок 8);

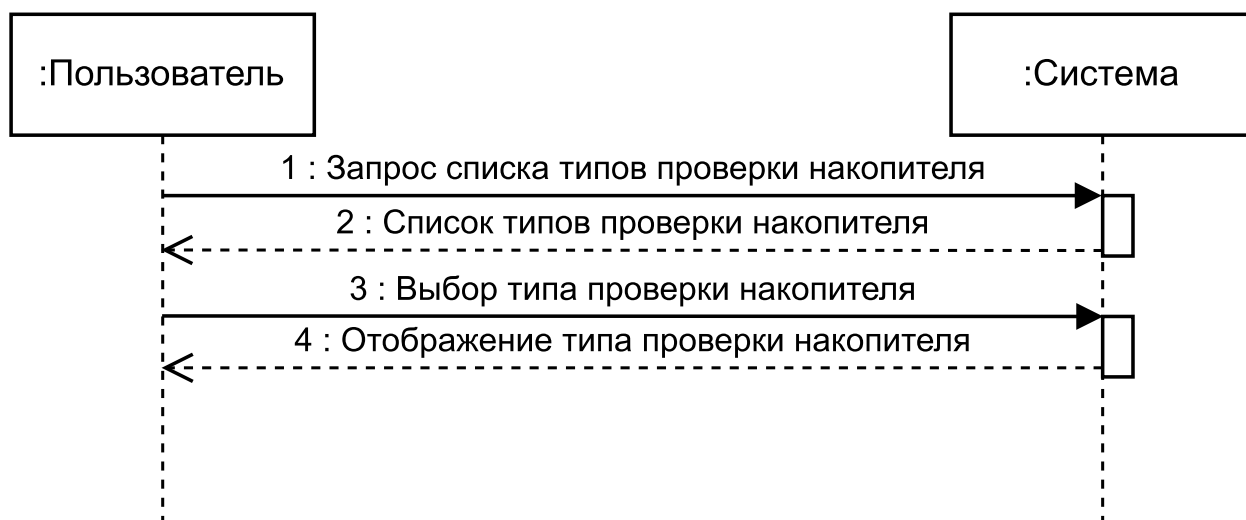


Рисунок 8 – Диаграмма последовательности действий для прецедента «Выбор типа проверки накопителя»

— для прецедента «Выбор типа записываемых данных» (рисунок 9);

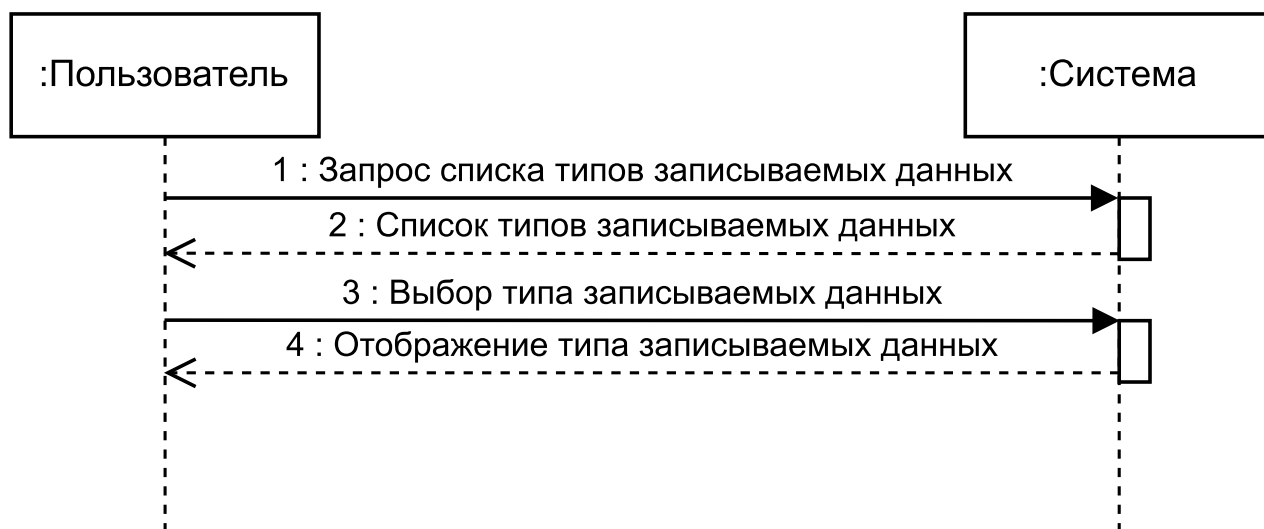


Рисунок 9 – Диаграмма последовательности действий для прецедента «Выбор типа записываемых данных»

— для прецедента «Тестирование накопителя» (рисунок 10).

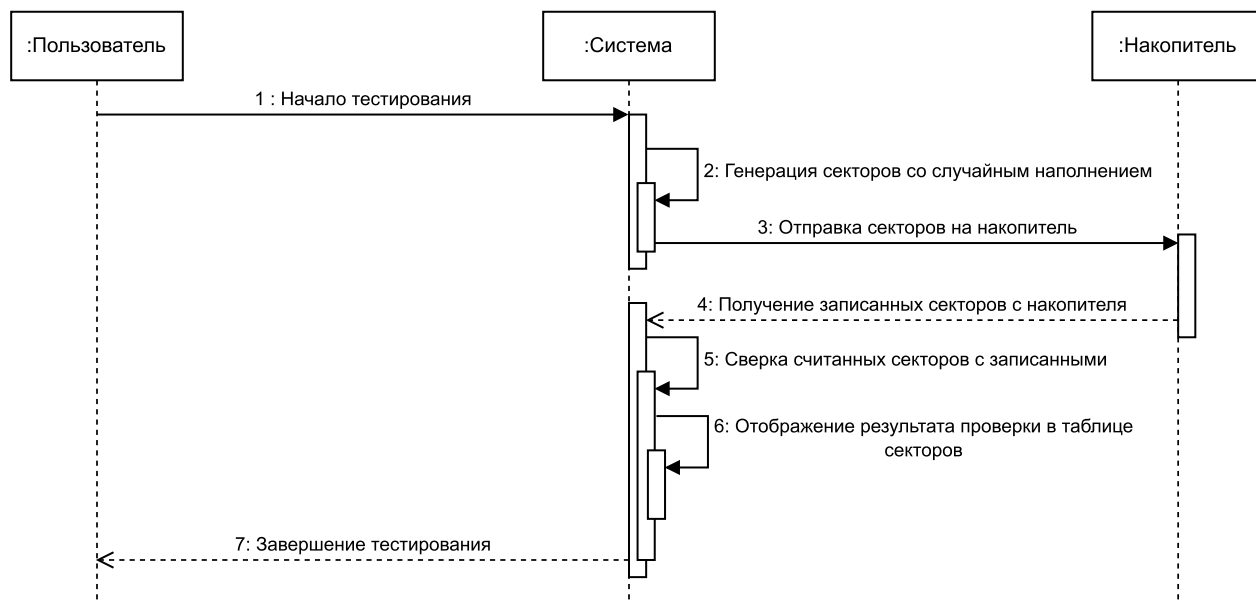


Рисунок 10 – Диаграмма последовательности действий для прецедента «Тестирование накопителя»

В ходе анализа вариантов использования были из проведенного сравнения аналогов были выявлены основные требования к приложению. Выявив основные требования, стало возможным составить диаграммы последовательности, которые конкретизируют действия, которые будут выполняться для вариантов использования приложения.

1.4 Техническое задание

5.1.1 Функциональные требования к программе

От разрабатываемого приложения ожидаются следующие характеристики и функции:

- возможность выбора тестируемого накопителя;
- возможность просмотра сведений о тестируемом накопителе (марка, модель, объем);
- возможность выбора типа проверки (деструктивная или недеструктивная);
- возможность выбора типа записываемых данных на запоминающее устройство (случайные или выбранные пользователем);
- возможность просмотра скорости записи данных (текущей и средней);
- возможность просмотра оценки оставшегося времени записи данных;
- возможность просмотра скорости чтения данных (текущей и средней);
- возможность оценки оставшегося времени чтения данных.

5.1.2 Требования к дизайну приложения

Приложение должно предоставлять возможность для редактирования параметров тестирования, таких как выбранный накопитель, тип тестирования, выбранные данные для тестирования на главной форме. Также пользователь должен иметь возможность увидеть на главной форме информацию о тестируемом накопителе. Во время тестирования пользователь не должен иметь возможности изменять параметры тестирования.

Во время тестирования главная форма приложения должна отображать карту объема накопителя в виде таблицы блоков. При наличии поврежденного сектора блок должен помечаться в наглядной для пользователя форме.

Дополнительные настройки, при наличии, должны быть вынесены в отдельное окно, которое пользователь может открыть при желании.

5.1.3 Требования к дизайну приложения

В качестве тестируемого накопителя для деструктивной проверки может выступать любое запоминающее устройство с возможностью записи на него данных подключенное по любой шине, поддерживаемой операционной системой пользовательского устройства: USB-накопитель, жесткий диск, твердотельный накопитель, карта памяти и т. д.

При использовании недеструктивной проверки для тестирования становятся доступны накопители без возможности записи на них данных, например CD, DVD, Blu-ray диски.

5.1.4 Требования к программному обеспечению

Для того чтобы запустить приложение потребуется:

- операционная система Windows 10 версии 1809 или более поздняя версия;
- Microsoft Visual C++ 2015-2022 Redistributable.

Данные требования следуют из требований к программному обеспечению разработанному с применением фреймворка Qt 6.5 [9].

5.1.5 Требования к дизайну приложения

Для того чтобы запустить приложение потребуется следующее аппаратное обеспечение:

- процессор: 64-х битный, 1 гигагерц (ГГц) или более быстрый процессор или система на чипе;
- оперативная память: 2 гигабайта (ГБ);
- объем накопителя: 20 гигабайт (ГБ);

– графический процессор: DirectX 9 или позже вместе с WDDM 1.0 драйвером;

– дисплей: 800 x 600 пикселей.

Требования основаны на требованиях для 64-х битной версии Windows 10 [10].

1.5 Техническое задание

В рамках данного раздела будут рассмотрены связки из языка программирования и фреймворка для разработки приложений с графическим интерфейсом пользователя. Будут рассмотрены преимущества и недостатки выбора связки из языка программирования и фреймворка и выбран лучший вариант.

Qt и C++ являются мощными инструментами для разработки кроссплатформенных приложений. Они предлагают широкий спектр возможностей для разработки графического интерфейса и позволяют легко работать с файловой системой и другими системными ресурсами. C++ позволяет работать с низкоуровневыми API операционной системы Windows, что особенно важно в случае приложений, которые работают с внешними накопителями [11, 12].

C# и Windows Forms, а также WPF – это технологии, разрабатываемые Microsoft для создания Windows-приложений. Они предоставляют широкий спектр инструментов для разработки графического интерфейса. Несмотря на то, что C# позволяет работать с низкоуровневыми API Windows, использовать эти API сложнее, так как это требует импортирования символов из библиотек. В C++ напротив, есть возможность использовать заголовочные файлы с определениями функций API Windows, что увеличивает скорость разработки приложения [13, 14].

JavaScript и Electron – мощные инструменты для разработки приложений с графическим интерфейсом. JavaScript является интерпретируемым языком программирования, что может существенно снизить производительность

приложения. Кроме того, Electron – это фреймворк, который основан на Chromium и Node.js, что может существенно увеличить размер приложения и потребление ресурсов компьютера. Так же, использовать язык JavaScript для работы с низкоуровневыми API операционной системы не представляется возможным [15].

После сравнения приведенных выше решений было принято решение использовать язык программирования C++ и фреймворк Qt.

C++ – это высокоуровневый язык программирования, который является обобщенной версией языка C. Он обладает мощной системой типов, поддерживает процедурное, объектно-ориентированное и обобщенное программирование. C++ также известен своей высокой производительностью и широкими возможностями в области системного программирования, создания приложений для научных вычислений и игр. Он является одним из самых популярных языков программирования и используется во многих приложениях и операционных системах [12].

Фреймворк Qt – это кроссплатформенный инструментарий разработки приложений на C++, который облегчает создание графических интерфейсов и многопоточных приложений. Он предоставляет широкий набор инструментов для работы с графикой, базами данных, сетью, мультимедиа и другими технологиями. Qt имеет простой и интуитивно понятный API, который позволяет разработчикам быстро создавать качественные и производительные приложения для разных платформ, включая Windows, macOS, Linux, Android и iOS. Благодаря своей эффективности и удобству использования, Qt стал одним из самых популярных фреймворков для разработки на C++ [11].

Устройство: [E:] KINGSTON DATATRavelER EXODIA 32GB ▾

Процент заполнения: 35%

Деструктивный режим: ☒ Данные для заполнения: Случайные ▾

Карта блоков	Данные о накопителе
<p>Марка: KINGSTON Модель: DATATRavelER EXODIA Объем: 8 Гб Средняя скорость чтения: 7 МБ/с Средняя скорость записи: 3 МБ/с</p>	

Скорость чтения: 8 МБ/с Скорость записи: 4 МБ/с

Начать проверку

Рисунок 12 – Макет главного окна приложения
с выбранной вкладкой «Данные о накопителе»

Данный макет интерфейса предполагается использовать с минимальными исправлениями. В финальном варианте приложения могут быть несовпадения с макетом.

2.2 Проектирование программных средств

При проектировании было принято решение разбить приложение на 6 основных модулей (рисунок 13):

- модуль DataProviders – этот модуль содержит в себе все классы, которые ответственны за предоставление данных для тестирования;
- модуль DeviceProviders – это модуль содержит в себе классы, ответственные за предоставление объектов устройств, с использованием которых в дальнейшем будет проводиться тестирование накопителей;
- модуль Devices – этот модуль содержит в себе классы, представляющие собой устройства, которые будут использоваться для тестирования, в данный

момент содержит только WindowsDevice, однако без проблем может быть расширен устройствами для других операционных систем;

- модуль System – этот модуль предоставляет в виде классов основные ресурсы операционной системы, такие как дескрипторы устройств, средства перечисления устройств и томов, обработку ошибок;

- модуль Widgets – этот модуль содержит в себе классы, представляющие собой графические элементы интерфейса пользователя, такие как основное окно программы или таблицу блоков устройства;

- модуль Services – этот модуль содержит классы, предоставляющие возможность выполнять тестирование накопителя в отдельном потоке исполнения.

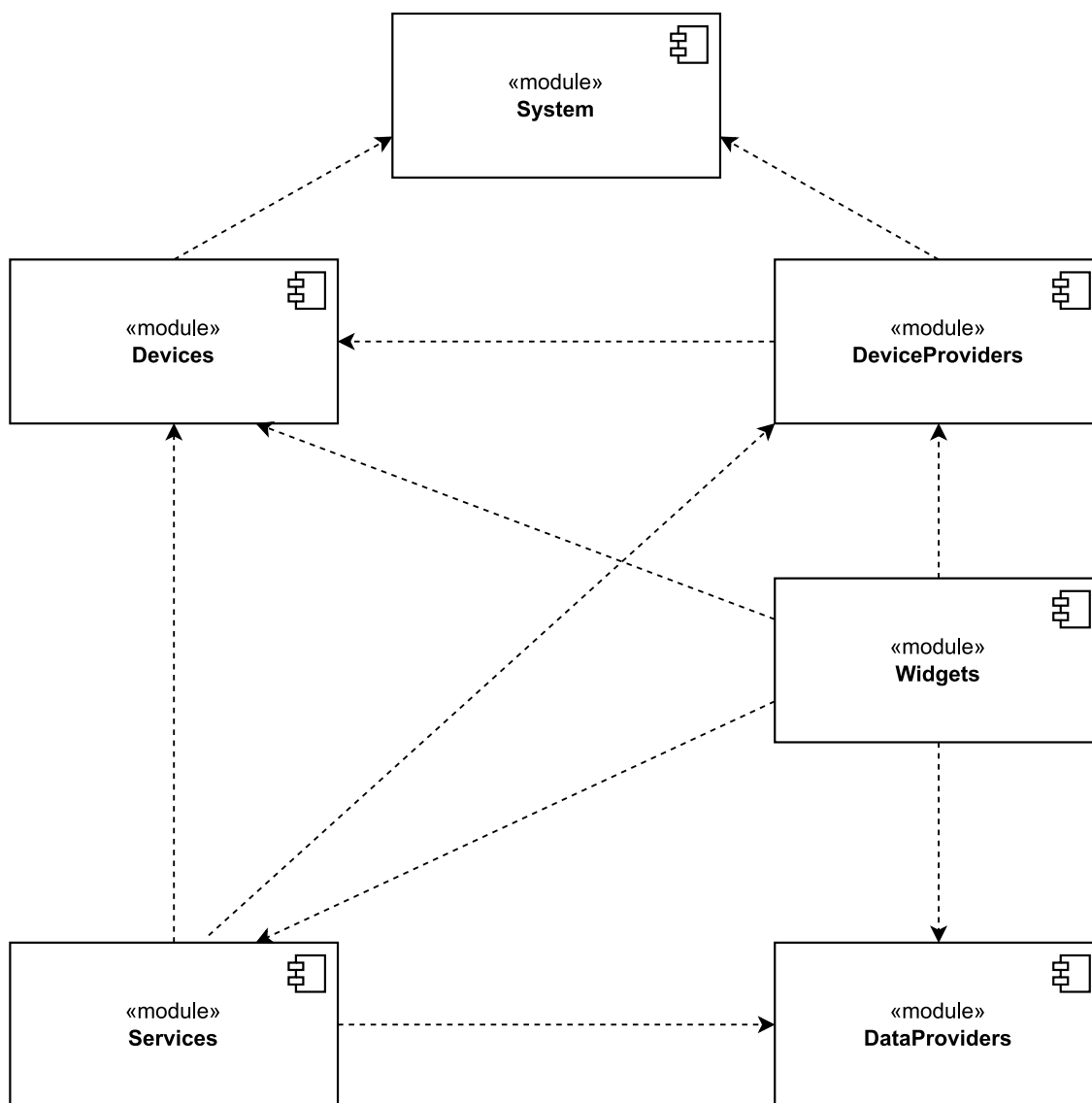


Рисунок 13 – Диаграмма компонентов приложения

Каждый из модулей содержит в себе классы, предоставляющие определенную функциональность внутри приложения, которая упрощает процесс кодирования. Рассмотрим структур классов приложения на примере модуля Services (рисунок 14):

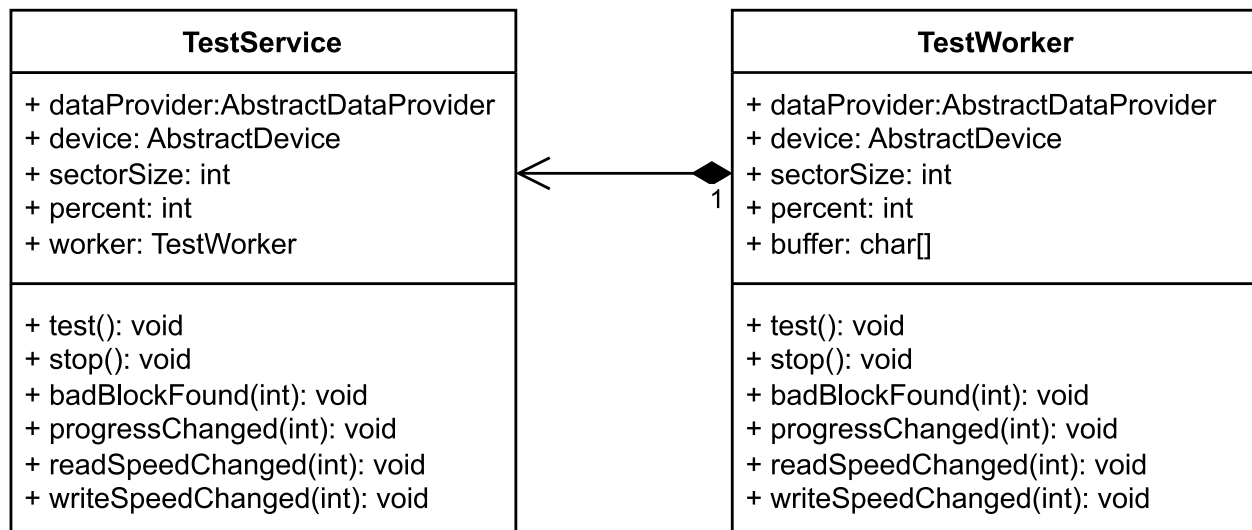


Рисунок 14 – Диаграмма классов модуля Services

Далее в тексте перечисляются спецификации классов из диаграммы на рисунке 14.

Класс TestService является сервисом для тестирования накопителя. TestService содержит в себе следующие члены:

- deviceProvider – объект для получения объектов устройств;
- device – устройство для тестирования;
- sectorSize – размер сектора тестируемого устройства;
- percent – процент от объема тестируемого накопителя;
- worker – объект, реализующий логику тестирования накопителя.

Функции класса TestService:

- test() – запустить тестирование накопителя;
- stop() – остановить тестирование накопителя;
- badBlockFound(int) – сигнал о том, что найден плохой блок;
- progressChanged(int) – сигнал об изменении прогресса тестирования;

– readSpeedChanged(int) – сигнал об изменении скорости чтения данных с накопителя;

– writeSpeedChanged(int) – сигнал об изменении скорости записи данных на накопитель.

Класс TestWorker является классом, реализующим тестирование накопителя. TestWorker содержит в себе следующие члены класса:

– deviceProvider – объект для получения объектов устройств;

– device – устройство для тестирования;

– sectorSize – размер сектора тестируемого устройства;

– percent – процент от объема тестируемого накопителя;

– buffer – буфер, содержащий данные для тестирования.

Функции класса TestWorker:

– test() – запустить тестирование накопителя;

– stop() – остановить тестирование накопителя;

– badBlockFound(int) – сигнал о том, что найден плохой блок;

– progressChanged(int) – сигнал об изменении прогресса тестирования;

– readSpeedChanged(int) – сигнал об изменении скорости чтения данных с накопителя;

– writeSpeedChanged(int) – сигнал об изменении скорости записи данных на накопитель.

В ходе проектирования класса TestWorker был спроектирован алгоритм тестирования накопителя.

Данный алгоритм позволяет проверить накопитель на наличие плохих блоков. Проверка производится путем записи тестовых данных, чтения записанных данных и их сверки с тестовыми данными.

Диаграмма деятельности для алгоритма тестирования накопителя находится на рисунке 15.

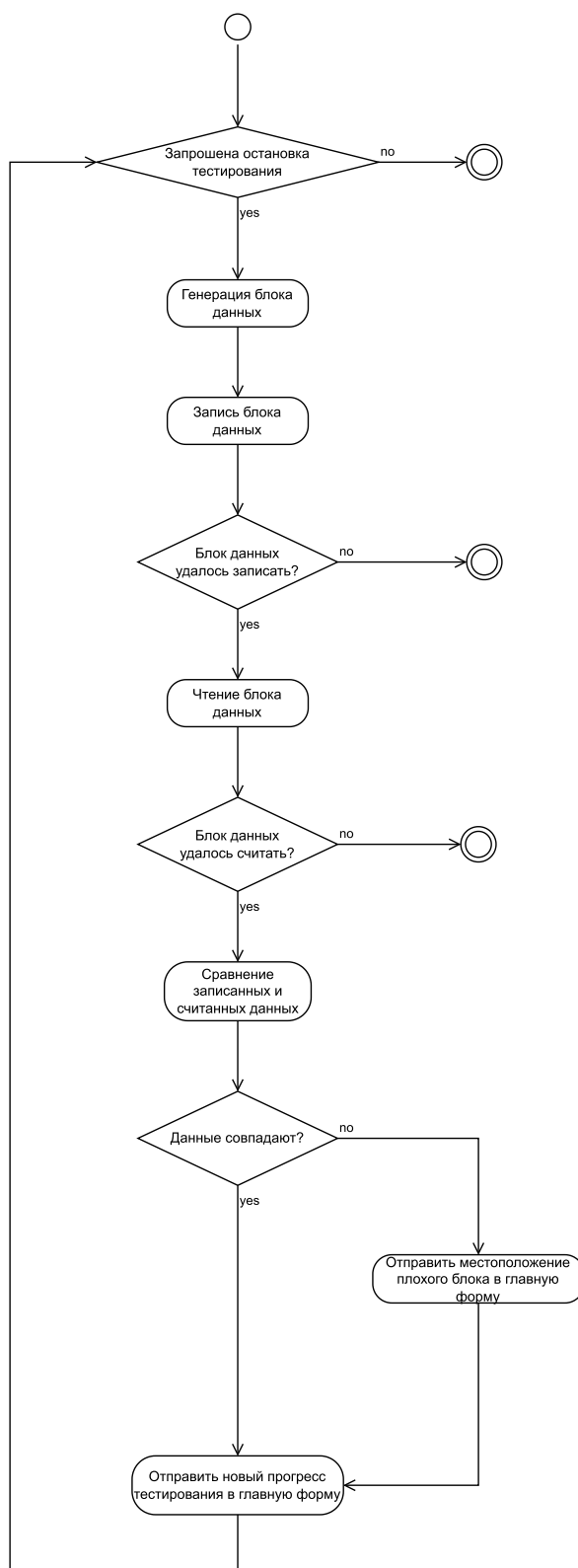


Рисунок 15 – Диаграмма деятельности для алгоритма тестирования накопителя

В ходе выполнения данной главы было произведено проектирование приложения: была спроектирована архитектура приложения и его графический интерфейс.

3 Планирование разработки и оценка бюджета

3.1 Календарное и ресурсное планирование

Планирование является неотъемлемой частью проектирования и разработки автоматизированных систем. Оно позволяет осуществлять управление и контроль за ходом работ по созданию информационной системы. Календарное планирование проекта включает в себя определение длительности работ и их взаимосвязей. С помощью диаграммы Ганта из приложения Т можно удобно представить сроки работ, продолжительность этапов, список требуемых ресурсов и другую информацию.

Календарное планирование также включает составление ресурсной модели проекта. Ресурсы могут быть трудовыми, материальными или невозобновляемыми, и они влияют на затраты проекта. Планирование ресурсов позволяет оптимизировать использование ресурсов и минимизировать их затраты [16].

Для составления графика проекта с оптимальным использованием ресурсов и минимальными сроками необходимо провести календарное планирование. Необходимо составить план стадий и этапов создания веб-приложения. В этом плане нужно указать наименование и длительность каждой стадии и этапа разработки, а также определить лиц, которые будут выполнять работы.

Важно представить план проекта в ясной и понятной форме, чтобы каждый участник мог понимать свои задачи и сроки их выполнения. Также следует учитывать риски и возможные изменения в процессе разработки. Управление рисками и управление изменениями являются неотъемлемой частью проектного менеджмента. Все эти аспекты важны для успешной разработки проекта.

Для составления календарного плана будет использоваться программа Microsoft Project.

В рамках проведения ресурсного планирования был составлен лист ресурсов проекта (рисунок 16).

Название ресурса	Тип	Единицы измерения материалов	Краткое название	Группа	Макс. единиц	Стандартная ставка	Ставка сверхурочных
Руководитель проекта	Трудовой		РП		100%	790,00 Р/ч	0,00 Р/ч
Программист (fullstack)	Трудовой		П		100%	620,00 Р/ч	0,00 Р/ч
Тестировщик	Трудовой		Т		100%	310,00 Р/ч	0,00 Р/ч
Дизайнер	Трудовой		Д		100%	620,00 Р/ч	0,00 Р/ч
Компьютер	Материальный		К			35 000,00 Р	
Принтер	Материальный		ПР			50 000,00 Р	
Картриджи для принтера	Материальный		КТ			1 500,00 Р	
Бумага	Материальный		Б			300,00 Р	
Интернет	Материальный		И			1 000,00 Р	
Технический писатель	Трудовой		ТП		100%	420,00 Р/ч	0,00 Р/ч

Рисунок 16 – Лист ресурсов проекта

После анализа этапов и их продолжительности возможно сделать вывод, что на разработку приложения потребуется 392 часа.

3.2 Расчет бюджета

Для расчета стоимости проекта необходимо учесть различные расходы, такие как амортизационные отчисления, материалы, зарплаты работников, социальные выплаты и другие расходы.

Амортизация – это процесс распределения стоимости производственных активов на стоимость готовой продукции. Для расчета амортизационных отчислений используется таблица 2, которая включает в себя срок полезного использования (СПИ), норму амортизации (НА), амортизацию (А), а также транспортно-заготовительные расходы (ТЗР).

Таблица 2 – Амортизационные отчисления

Оборудование	Стоимость, руб.	СПИ, лет	НА, %	А, год, руб.
Компьютер	35 000	5	20	7 000
Принтер	45 000	10	10	4500
ТЗР (1%)	1000			
Итого	246 000			24 500

Для разработки приложения будут использоваться следующие программные продукты: Qt Creator, Visual Studio 2022, фреймворк Qt.

Амортизационные отчисления рассчитываются по следующей формуле:

$$A = C_{\text{пер}} * НА,$$

где А амортизационные начисления, руб.;

$C_{\text{пер}}$ – первоначальная стоимость основных средств, руб.

НА – норма амортизации, которая находится по формуле:

$$НА = \frac{1}{\text{СПИ}},$$

где СПИ – срок полезного использования основных средств.

Срок полезного использования компьютера = 5 лет,

$$НА_{\text{комп}} = \frac{1}{5} = 0,2.$$

Срок полезного использования принтера = 10 лет,

$$НА_{\text{принт}} = \frac{1}{10} = 0,1,$$

$$A_{\text{комп}} = 200000 * 0,2 = 40000 \text{ руб.},$$

$$A_{\text{принт}} = 45000 * 0,1 = 4500 \text{ руб.},$$

$$A_r = 40000 + 4500 + 1000 = 45500 \text{ руб.},$$

$$A_m = \frac{45500}{12} = 3791,66 \text{ руб./мес.},$$

$$A_{\text{итог}} = 3791,66 * 1 = 3791,66 \text{ руб.}$$

Работу по созданию приложения предполагается вести 392 часов, расчёт будет проведен за 1 месяц.

Таблица 3 содержит расчеты оборотных средств, которые были использованы при разработке программного продукта.

Таблица 3 – Затраты по статье «Материалы»

Наименование	Количество, шт.	Сумма, руб.
Бумага	2	600
ТЗР (1%)		10
ИТОГО:		606

Зарплата – вознаграждение за труд в зависимости от квалификации работника, сложности, количества, качества и условий выполняемой работы.

Таблица 4 содержит информацию о трудозатратах всех типов работ, связанных с разработкой данного программного продукта.

Таблица 4 – Расчет основной и дополнительной заработной платы

Исполнитель	Трудоемкость, часов (Т _р)	Ставка, руб./час	ОЗП, руб.	ДЗП, руб.	Соц. Выплаты
Руководитель проекта	108	790	85 514,13	8 551,41	28 219,66
Программист (fullstack)	159	620	98 154,97	9 815,50	32 391,14
Тестировщик	61	310	16 196,01	1 619,60	5 344,68
Дизайнер	8	620	4 953,81	495,38	1 634,76
Технический писатель	56	420	23 307,27	2 330,73	7 691,40
ИТОГО:	392		228 126,19	22 812,62	75 281,64

На весь процесс разработки приложения затрачено 392 часа. Рассчитаем оплату труда руководителя проекта.

Оплата за труд рассчитывается следующим образом:

$$\text{ОЗП} = C_{\text{ч}} * T_{\text{р}},$$

где $T_{\text{р}}$ – трудоемкость;

$C_{\text{ч}}$ – тарифная ставка, которая рассчитывается по формуле:

$$C_{\text{ч}} = \frac{\text{МРОТ} * T_{\text{к}}}{160},$$

где МРОТ – минимальный размер оплаты труда = 12 130 руб.;

$T_{\text{к}}$ – тарифный коэффициент;

160 – количество часов в месяц по ТК РФ.

$$T_{\text{кррук}} = 6,8,$$

$$C_{\text{чрук}} = \frac{16242 * 7,8}{160} = 791,80,$$

$$\text{ОЗП}_{\text{рук}} = 791,80 * 108 = 85 514,13 \text{ руб.}$$

Мотивация труда рассчитывается через дополнительную заработную плату:

$$\text{ДЗП} = 10\% \text{ ОЗП},$$

$$\text{ДЗП}_{\text{рук}} = 10\% * 85514,13 = 8551,41 \text{ руб.}$$

Единый тариф страховых взносов – консолидированный единый тариф страховых взносов на обязательное пенсионное страхование, обязательное медицинское страхование и социальное страхование на случай временной

нетрудоспособности и в связи с материнством. Составляет 30% от заработной платы до достижения предельной величины базы.

$$СВ = 30\% * (ОЗП + ДЗП),$$

$$СВ_{рук} = 30\% * (85514,13 + 8551,41) = 28219,66 \text{ руб.}$$

В этой статье приведены затраты на доступ в сеть Интернет и Электроэнергию (ЭЭ). ЭЭ рассчитывается по формуле:

$$ЭЭ = T_c * W * T_{раб},$$

где T_c = стоимость 1 кВт*ч = 2,97 руб;

W – энергомощность компьютера = 750 Вт = 0,75 кВт;

$T_{раб}$ – время создания, т. к. программный продукт разрабатывался 392 часов.

$$ЭЭ = 2,97 * 0,75 * 392 = 873,18 \text{ руб.}$$

Стоимость услуги доступа в Интернет:

Работа над программным продуктом велась 392 часа, тариф за услугу доступ в Интернет составляет = 1000 руб./мес. Так как месяц пользования интернетом предоплачивается полностью, то расчет стоимости пользования ведется за 2 месяца:

$$\text{Интернет} = 1000 * 2 = 2000 \text{ руб.},$$

$$\text{Прочие затраты} = ЭЭ + \text{Интернет} = 873,18 + 2000 = 2873,18 \text{ руб.}$$

Структура затрат представлена в таблице 5.

Таблица 5 – Структура затрат

Наименование статей	Затраты, руб.	%
Амортизация	3 791,66	1,13
Материалы	606	0,18
Фонд основной и дополнительной ЗП	250 938,81	74,83
Социальные выплаты	75 281,64	22,45
Прочие	4 721,64	1,41
Затраты на проект	335 339,75	

Диаграмма структуры затрат представлена на рисунке 17.

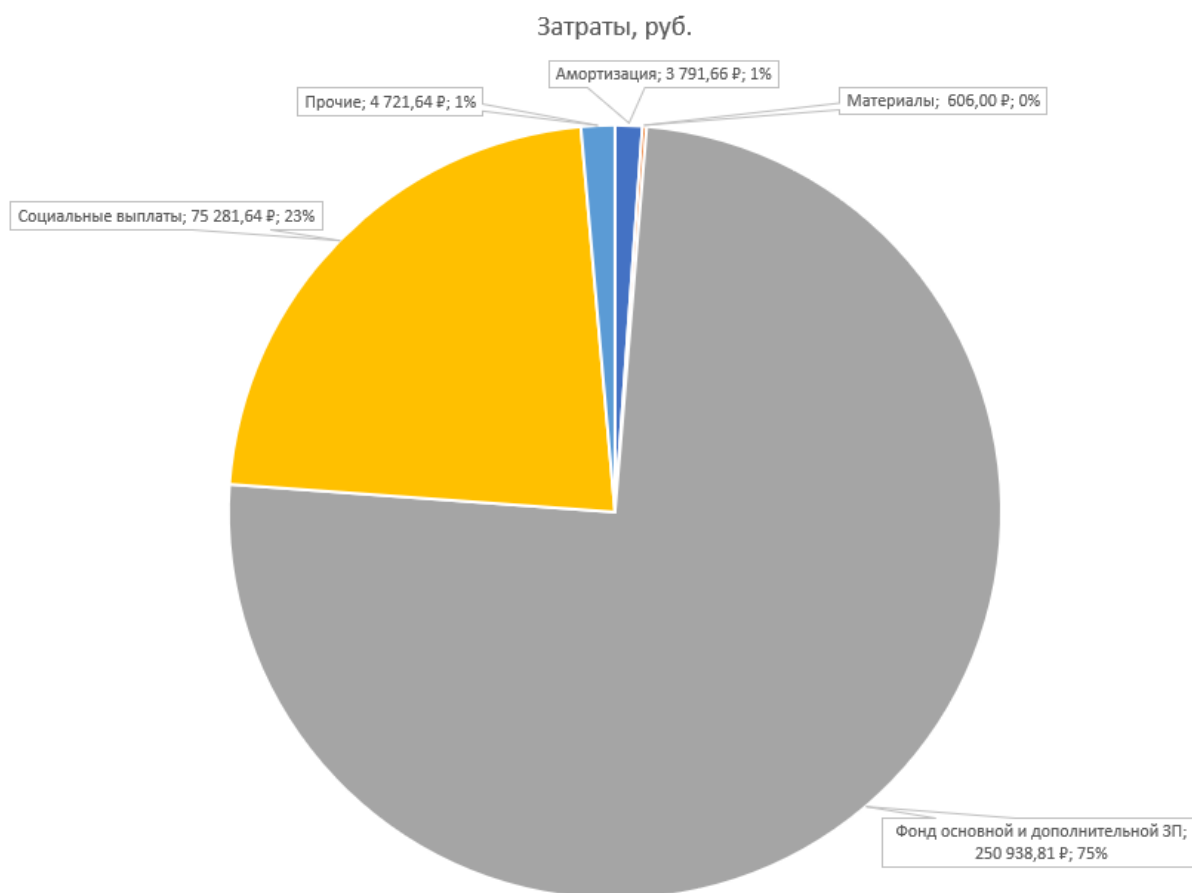


Рисунок 17 – Структура затрат

После анализа структуры расходов можно установить, что основные статьи расходов – это фонд оплаты труда и социальные выплаты. Стоимость всего процесса разработки программного продукта «Программа для проверки внешних накопителей» составляет 335 339,75 рублей.

4 Реализация приложения

4.1 Кодирование приложения

В ходе проектирования приложение было разбито на 6 модулей, что наложило определенные требования на структуру файлов проекта – проект состоит из 6 папок, каждая из которых содержит реализацию только тех классов, что относятся к модулю, который представляется папкой. Такая структура проекта упрощает процесс кодирования и позволяет группировать файлы с программным кодом.

Рассмотрим содержимое каждой из папок кодируемого приложения.

Папка `dataproviders` – папка, содержащая код классов, отвечающих за генерацию данных для тестирования. Папка содержит следующие файлы:

- `abstractdataprovider.h` – файл, содержащий определение класса `AbstractDataProvider`, ответственного за интерфейс взаимодействия с объектами, предоставляющими данные для тестирования;

- `abstractdataprovider.cpp` – файл, содержащий реализацию класса `AbstractDataProvider`;

- `randomdataprovider.h` – файл, содержащий определение класса `RandomDataProvider`, предоставляющего случайные данные для тестирования;

- `randomdataprovider.cpp` – файл, содержащий реализацию класса `RandomDataProvider`;

- `templatedataprovider.h` – файл, содержащий определение класса `TemplateDataProvider`, предоставляющего шаблонные данные для тестирования;

- `templatedataprovider.cpp` – файл, содержащий реализацию класса `TemplateDataProvider`.

Папка `deviceproviders` – папка, содержащая код классов, отвечающих за получение устройств для тестирования. Папка содержит следующие файлы:

- `abstractdeviceprovider.h` – файл, содержащий определение класса `AbstractDeviceProvider`, ответственного за интерфейс взаимодействия с объектами, предоставляющими устройства для тестирования;

- `abstractdeviceprovider.cpp` – файл, содержащий реализацию класса `AbstractDeviceProvider`;

- `windowsdeviceprovider.h` – файл, содержащий определение класса `WindowsDataProvider`, предоставляющего устройства для тестирования на операционной системе Windows;

- `windowsdeviceprovider.cpp` – файл, содержащий реализацию класса `WindowsDataProvider`.

Папка `devices` – папка, содержащая код классов, отвечающих за получение устройств для тестирования. Папка содержит следующие файлы:

- `abstractdevice.h` – файл, содержащий определение класса `AbstractDevice`, ответственного за интерфейс взаимодействия с объектами, предоставляющими интерфейс для взаимодействия с тестируемыми устройствами;

- `abstractdevice.cpp` – файл, содержащий реализацию класса `AbstractDevice`;

- `windowsdevice.h` – файл, содержащий определение класса `WindowDevice`, предоставляющего интерфейс для взаимодействия с устройствами ОС Windows;

- `windowsdevice.cpp` – файл, содержащий реализацию класса `WindowsDevice`.

Папка `services` – папка, содержащая код классов, отвечающих за получение устройств для тестирования. Папка содержит следующие файлы:

- `testservice.h` – файл, содержащий определение класса `TestService`, предоставляющего возможность запуска тестирования в параллельном потоке;

- `testservice.cpp` – файл, содержащий реализацию класса `TestService`;

- `testworker.h` – файл, содержащий определение класса `TestWorker`, предоставляющего методы для проведения тестирования накопителя;

- `testworker.cpp` – файл, содержащий реализацию класса `TestWorker`.

Папка `system` – папка, содержащая код классов, отвечающих за взаимодействие с операционной системой Windows. Папка содержит следующие файлы:

- `diskdevices.h` – файл, содержащий определение класса `DiskDevices`, предоставляющего возможность получения дескрипторов дисковых устройств;
- `diskdevices.cpp` – файл, содержащий реализацию класса `DiskDevices`;
- `handle.h` – файл, содержащий определение класса `Handle`, предоставляющего методы для взаимодействия с дескриптором устройства;
- `handle.cpp` – файл, содержащий реализацию класса `Handle`;
- `volumesearchhandle.h` – файл, содержащий определение класса `VolumeSearchHandle`, предоставляющего методы для получения дескрипторов томов;
- `volumesearchhandle.cpp` – файл, содержащий реализацию класса `VolumeSearchHandle`;
- `errorhandling.h` – файл, содержащий определения функций, позволяющих обработать ошибки Windows API;
- `errorhandling.cpp` – файл, содержащий реализацию функций из `errorhandling.h`.

Папка `widgets` – папка, содержащая код классов, реализующий Qt-виджеты приложения. Папка содержит следующие файлы:

- `abstractmainwindow.h` – файл, содержащий определение класса `AbstractMainWindow`, предоставляющего интерфейс для взаимодействия с главной формой приложения;
- `abstractmainwindow.cpp` – файл, содержащий реализацию класса `AbstractMainWindow`;
- `blockview.h` – файл, содержащий определение класса `BlockView`, предоставляющего графический интерфейс пользователя, отображающий карту блоков накопителя;
- `blockview.cpp` – файл, содержащий реализацию класса `BlockView`;

Листинг 1 – Программный код функции тестирования.

```

void TestWorker::test()
{
    device()->setPosition(device()->size() - device()-
>sectorSize());
    for (qsize_t iteration = 0;; iteration += 100)
    {
        for (qsize_t i = 0; i < percent(); ++i) {
            if (QThread::currentThread()-
>isInterruptionRequested() ||
                (writeChunk() && readChunk(i)))
            {
                return;
            }
            bool stop = writeChunk();
            stop |= readChunk(iteration);
            if (stop)
                return;
        }
        quint64 progress = (iteration + 100) * BLOCK_COUNT *
device()->sectorSize();
        emit progressChanged(progress);
        if (!device()->setPosition(progress))
        {
            return;
        }
    }
}

bool TestWorker::writeChunk()
{
    bool stop = false;
    dataProvider()->getDataBlock(deviceBuffer.data(), BLOCK_COUNT
* sectorSize());
    std::copy(deviceBuffer.data(), deviceBuffer.data() +
BLOCK_COUNT * sectorSize(), compareBuffer.data());
    quint64 speed = measureSpeed([this, &stop]() {
        for (qsize_t i = 0; i < BLOCK_COUNT; ++i) {
            qsize_t written = device()-
>writeBlock(deviceBuffer.data() + i * sectorSize(), sectorSize());
            if (written == 0) {
                stop = true;
                return i;
            }
        }
        return BLOCK_COUNT;
    });
    emit writeSpeedChanged(speed);
    return stop;
}

bool TestWorker::readChunk(qsize_t iteration)
{

```

```

    bool stop = false;
    device()->setPositionRelative(-1 * (BLOCK_COUNT *
sectorSize()));
    quint64 speed = measureSpeed([this, &stop, iteration]() {
        for (qsize_t i = 0; i < BLOCK_COUNT; ++i) {
            auto read = device()->readBlock(deviceBuffer.data() +
i * sectorSize(), sectorSize());
            if (read == 0) {
                stop = true;
                compareChunk(iteration, i);
                return i;
            }
        }
        compareChunk(iteration, BLOCK_COUNT);
        return BLOCK_COUNT;
    });
    emit readSpeedChanged(speed);
    return stop;
}

```

В ходе кодирования приложение было разработано приложение, соответствующее требованиям к архитектуре, разработанным в ходе этапа проектирования приложения.

5 Анализ качества программного продукта

5.1 Тестирование программного продукта

На первом этапе тестирования планируется воспользоваться методом чёрного ящика, который широко известен как функциональное тестирование. Данный метод позволяет изучить функциональные возможности программного обеспечения, не обладая детальными знаниями об его внутренней структуре кода. Метод чёрного ящика может применяться на всех уровнях тестирования, в особенности на высоких уровнях. Тестирование по стратегии чёрного ящика дает возможность исследовать поведение системы, не зависимо от её внутренней структуры, архитектуры и реализации [17].

5.1.1 Метод классов эквивалентности

Метод классов эквивалентности является одним из методов тестирования программного обеспечения, который применяется для уменьшения количества тестовых сценариев, не уменьшая при этом покрытие тестами. Он основывается на классификации входных данных на классы эквивалентности, где каждый класс представляет собой набор входных данных, который должен привести к одному и тому же поведению системы. После классификации для каждого класса выбирается один представитель для тестирования, что позволяет уменьшить количество необходимых тестовых сценариев (таблица 6) [18].

Таблица 6 – Табличное представление классов эквивалентности

	Правильные КЭ	Неправильные КЭ
Длина числа в символах	1-3 (1)	0 (2) 4 – ∞ (3)
Численное значение	1-100 (4)	0 (5), 101 – ∞ (6)
Содержание	Цифры (7)	Остальные символы (8)

Далее приведено графическое представление классов эквивалентности (рисунки 19 – 21).



Рисунок 19 – Графическое представление класса «Длина числа в символах»



Рисунок 20 – Графическое представление класса «Численное значение»

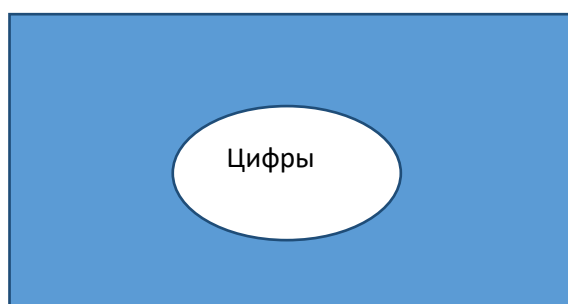


Рисунок 21 – Графическое представление класса «Содержание»

На основе приведенных выше классов эквивалентности были составлены тестовые сценарии (тест-кейсы). Составленные тестовые сценарии и их результаты представлены в таблице 7.

Таблица 7 – Тест-кейсы для классов эквивалентности

№	КЭ	Входные данные	Выходные данные
0	Все правильные 1,4,7	53	Успешно
1	1,5,7	0	Ошибка: процент заполнения диска не может быть нулевым
2	1,6,7	153	Ошибка: число больше 100
3	3,6,7	1111	Ошибка: число больше 100
4	1,8	abc	Ошибка: введите число
5	3,8	abcefgghi	Ошибка: введите число

В результате выявления классов эквивалентности были найдены тест-кейсы для тестируемого значения, а также было проведено тестирование найденных тест-кейсов.

5.1.2 Метод граничных условий

Метод граничных условий является техникой тестирования программного обеспечения, которая направлена на проверку корректности поведения программы на границах допустимых значений. Этот метод заключается в выборе тестовых данных, которые лежат на границах допустимых значений входных параметров, и проверке, как программа обрабатывает эти данные. Основная цель этого метода – выявление ошибок, которые могут возникнуть при обработке граничных значений [18].

Выделенные граничные условия приведены ниже в таблице 8.

Таблица 8 – Граничные условия

Границы	
Физические	Процент не является целым числом (Ф1)
Логические	Процент является отрицательным числом (Л1) Процент больше 100 (Л2)
Технологические	Процент больше максимально возможного по типу данных (Т1) Процент меньше минимально возможного по типу данных (Т2)
Произвольные	Процент не может быть нулевым (П1)

Для представленных выше граничных значений были составлены соответствующие тест-кейсы и их результаты (таблица 9).

Таблица 9 – Тест-кейсы для граничных условий

№	Границы	Входные данные	Выходные данные
6	Ф1	Float: percent = 12.12	Ошибка: тип данных не является целым числом
7	Л1	percent = -1	Ошибка: процент заполнения диска не может быть отрицательным
8	Л2	percent = 101	Ошибка: Число больше 100
9	Т1	percent = 9999999999999999	Ошибка: Число превышает максимально возможное значение для типа данных
10	Т2	percent = -9999999999999999	Ошибка: Число меньше, чем минимально возможное значение для типа данных
11	П1	percent = 0	Ошибка: процент заполнения диска не может быть нулевым

В ходе выполнения тестирования по методу граничных условий были выявлены тест-кейсы для граничных условий, а также было проведено тестирование в соответствии с найденными граничными условиями.

5.1.3 Функциональное тестирование

На основе выявленных требований к приложению был составлен план тестирования и результаты пунктов тестирования. Результаты функционального тестирования представлены в таблице 10.

Таблица 10 – Перечень тестов приложения

п/п	Назначение теста	Описание теста	Результат теста
1	Выбрать тестируемый накопитель	После выбора накопителя из выпадающего списка становится возможным использовать накопитель для тестирования.	Тест пройден (рисунок С.1).
2	Выбрать тип проверки накопителя	После установки флажка изменяется режим тестирования.	Тест пройден (рисунок С.2).
3	Выбрать тип записываемых данных	После выбора типа записываемых данных изменения отображаются на форме.	Тест пройден (рисунок С.3).
4	Протестировать накопитель	В ходе тестирования накопителя прогресс тестирования отображается на карте блоков.	Тест пройден (рисунок С.4).
5	Просмотреть сведения о тестируемом накопителе	После выбора вкладки с информацией о накопителе становится возможным просмотреть данные о накопителе.	Тест пройден (рисунок С.5).
6	Просмотреть оставшееся время тестирования	После начала тестирования становится возможным просмотреть оценку времени тестирования накопителя.	Тест пройден (рисунок С.6).

В ходе проведения функционального тестирования были проверены основные варианты использования приложения. Все варианты использования выполняются корректно, что говорит о том что приложение пригодно для использования в соответствии с этими вариантами использования.

5.2 Метрики кода

Метрикой SLOC является измерение количества строк исходного кода, включая комментарии [19].

Вычислим метрики SLOC по коду, который приведен в листинге 2.

Листинг 2 – Код для SLOC метрик.

```
#include "testservice.h"
#include "testworker.h"
TestService::TestService(QObject *parent)
    : QObject{parent}
{
    worker = new TestWorker();
    thread = new QThread(this);
    worker->moveToThread(thread);
    connect(this, &TestService::dataProviderChanged, worker,
&TestWorker::setdataProvider);
    connect(this, &TestService::deviceChanged, worker,
&TestWorker::setDevice);
    connect(this, &TestService::sectorSizeChanged, worker,
&TestWorker::setsectorSize);
    connect(this, &TestService::test, worker, &TestWorker::test);
    connect(this, &TestService::percentChanged, worker,
&TestWorker::setpercent);
    connect(worker, &TestWorker::badBlockFound, this,
&TestService::badBlockFound);
    connect(worker, &TestWorker::progressChanged, this,
&TestService::progressChanged);
    connect(worker, &TestWorker::readSpeedChanged, this,
&TestService::readSpeedChanged);
    connect(worker, &TestWorker::writeSpeedChanged, this,
&TestService::writeSpeedChanged);
    thread->start();
}
AbstractDataProvider *TestService::dataProvider() const
{
    return worker->dataProvider();
}
void TestService::setdataProvider(AbstractDataProvider
*newDataProvider)
{
    emit dataProviderChanged(newDataProvider);
}
AbstractDevice *TestService::device() const
{
    return worker->device();
}
void TestService::setDevice(AbstractDevice *newDevice)
{
    emit deviceChanged(newDevice);
}
qsize_t TestService::sectorSize() const
{
    return worker->sectorSize();
}
```

```

void TestService::stop()
{
    thread->requestInterruption();
}
void TestService::setsectorSize(qsizetype newSectorSize)
{
    emit sectorSizeChanged(newSectorSize);
}
int TestService::percent() const
{
    return worker->percent();
}
void TestService::setpercent(int newPercent)
{
    emit percentChanged(newPercent);
}

```

В данной программе имеем:

- 67 физических строк кода;
- 35 логических строк кода;
- 0 строк комментариев.

5.2.1 Цикломатическая сложность

Проведем вычисление метрик цикломатической сложности. Для этого возьмём код проверки накопителя, который был вынесен в отдельную функцию. Блок рассматриваемого кода приведен листинге 3. Граф приведен в приложении У.

Листинг 3 – Код для вычисления цикломатической сложности

```

void TestWorker::test()
{
    device()->setPosition(device()->size() - device()-
>sectorSize());
    for (qsizetype iteration = 0;; iteration += 100)
    {
        for (qsizetype i = 0; i < percent(); ++i) {
            if (QThread::currentThread()-
>isInterruptionRequested() ||
                (writeChunk() && readChunk(i)))
            {
                return;
            }
            bool stop = writeChunk();
            stop |= readChunk(iteration);
        }
    }
}

```

```

        if (stop)
            return;
    }
    quint64 progress = (iteration + 100) * BLOCK_COUNT *
device()->sectorSize();
    emit progressChanged(progress);
    if (!device()->setPosition(progress))
    {
        return;
    }
}
}

```

Для вычисления цикломатической сложности $V(G)$ применяется следующая формула:

$$V(G) = e - n + 2p,$$

где e – количество дуг, n – количество вершин, p – число компонент связности [20].

Граф содержит 14 вершин, 20 дуг и 0 компонент связности:

$$V(G) = 20 - 14 + 0 = 6.$$

Таким образом значение цикломатической сложности равно 6.

5.2.2 Метрики Холстеда

Приведем замеры по метрикам Холстеда, метрики будут реализованы на основе метода для авторизации. Ниже приведены таблицы с данными для данной метрики (таблицы 11, 12) и сам код данного метода (листинг 4).

Листинг 4 – Код для метрик Холстеда

```

bool TestWorker::writeChunk()
{
    bool stop = false;
    dataProvider()->getDataBlock(deviceBuffer.data(), BLOCK_COUNT
* sectorSize());
    std::copy(deviceBuffer.data(), deviceBuffer.data() +
BLOCK_COUNT * sectorSize(), compareBuffer.data());
    quint64 speed = measureSpeed([this, &stop]() {
        for (qsize_t i = 0; i < BLOCK_COUNT; ++i) {
            qsize_t written = device()-
>writeBlock(deviceBuffer.data() + i * sectorSize(), sectorSize());

            if (written == 0) {
                stop = true;
            }
        }
    });
}

```

```

        return i;
    }
}
return BLOCK_COUNT;
});
emit writeSpeedChanged(speed);
return stop;
}

```

Таблица 11 – Операторы

Операторы	Число вхождений
.	5
->	2
+	2
*	3
,	5
=	2
==	1
<	1
++	1
return	3
;	12
emit	1

Таблица 12 – Операнды

Операнды	Число вхождений
false	1
stop	3
BLOCK_COUNT	4
written	2
true	1
i	5
speed	2
this	1

Длина программы рассчитывается по формуле:

$$N = N1 + N2 = 38 + 19 = 57.$$

Словарь программы рассчитывается по формуле:

$$n = n1 + n2 = 12 + 8 = 20.$$

Объем программы рассчитывается по формуле:

$$HV = N * \log_2(n) = 57 * 4,32 = 246,4.$$

Потенциальный объем программы:

$$V^* = (2 + n) * \log_2(2 + n) = 5 * \log_2(5) = 11,6.$$

Теоретический уровень программы рассчитывается по формуле:

$$L' = \frac{V^*}{HV} = \frac{11,6}{246,4} = 0,05.$$

Сложность программы рассчитывается по формуле:

$$HD = \left(\frac{n1}{2}\right) * \left(\frac{N2}{n2}\right) = \left(\frac{12}{2}\right) * \left(\frac{19}{8}\right) = 14,25.$$

В ходе вычисления метрик Холстеда были получены значения, по которым можно оценить сложность разработанного приложения. Значения не выходят за рамки нормы, что говорит о том, что сложность поддержки приложения не будет высокой.

Заключение

Результатом выполнения бакалаврской работы является спроектированная программа для проверки внешних накопителей.

В ходе выполнения этапа анализа требований были рассмотрены приложения-аналоги и осуществлен их анализ: были выделены преимущества и недостатки, а также функциональные особенности и возможности каждого из них. Также было проведено сравнение аналогов между собой. Результатом данных сравнений стали функциональные требования к проектируемому web-приложению. Также были выявлены дополнительные функциональные требования, проведен их анализ.

В ходе выполнения этапа анализа требований была обоснована актуальность разрабатываемого приложения. Рассмотрены приложения-аналоги и осуществлен их обзор, выделена основная функциональность, которую способен выполнять каждый из аналогов. После рассмотрения функциональности было принято решение о реализации функциональностей, присутствующих в приложениях аналогичного назначения.

В ходе выполнения этапа проектирования приложения определена архитектура приложения. Были определены основные модули, из которых будет состоять приложение, а также классы, входящие в модули приложения.

Был составлен план и проведена оценка бюджета. В ходе выполнения данного этапа выявлены временные и финансовые затраты на разработку и необходимые ресурсы. Результатом стала диаграмма Ганта и отчет. После анализа структуры расходов удалось установить, что основные статьи расходов – это фонд оплаты труда и социальные выплаты. Стоимость всего процесса разработки программного продукта составила 335 339,75 рублей.

Спроектированное приложение было реализовано на языке C++ с помощью платформы для разработки графических приложений Qt 6. Были составлены диаграммы классов и компонентов, для алгоритмов были составлены диаграммы последовательностей.

Было проведено функциональное тестирование разработанного приложения, что позволило определить корректность функционирования программного продукта. Кроме того, были вычислены метрики кода, позволившие оценить сложность и объем программного кода продукта.

В результате выполнения работы все поставленные задачи были выполнены.

Список использованных источников

1. What Does Bad Sector Mean? // Technopedia [Электронный ресурс]. URL: <https://www.techopedia.com/definition/992/bad-sector> (дата обращения: 24.05.2023).
2. What Does Storage Device Mean? // Technopedia [Электронный ресурс]. URL: <https://www.techopedia.com/definition/1119/storage-device> (дата обращения: 24.05.2023).
3. Fkrezgy, Yohannes. Solid state drive (SSD) digital forensics construction. // Fkrezgy Yohannes, М.: Politecnico di Milano., 2010 – 59 с.
4. Henderson, Harry. Encyclopedia of computer science and technology / Harry Henderson. – Rev. ed., М.: Facts On File., 2009 – 593 с.
5. badblocks – search a device for bad blocks // Ubuntu Manpages [Электронный ресурс]. URL: <https://manpages.ubuntu.com/manpages/jammy/en/man8/badblocks.8.html> (дата обращения: 24.05.2023).
6. H2testw 1.4 // Heise [Электронный ресурс]. URL: <https://www.heise.de/download/product/h2testw-50539> (дата обращения: 24.05.2023).
7. Bulldog – disclose the fraud // Bulldog [Электронный ресурс]. URL: <https://disktest.ru> (дата обращения: 24.05.2023).
8. Flashnul Readme // GitHub [Электронный ресурс]. URL: <https://github.com/amaraoflashnul/blob/master/readme.rus.html> (дата обращения: 24.05.2023).
9. Supported Platforms // Qt [Электронный ресурс]. URL: <https://doc.qt.io/qt-6/supported-platforms.html> (дата обращения: 24.05.2023).
10. Windows 10 system requirements // Microsoft [Электронный ресурс]. URL: <https://support.microsoft.com/en-us/windows/windows-10-system-requirements-6d4e9a79-66bf-7950-467c-795cf0386715> (дата обращения: 24.05.2023).
11. Шлее, М. Qt 4.8. Профессиональное программирование на C++. / Шлее М. – СПб.: БХВ-Петербург, 2012. — 912 с.

12. Stroustrup, Bjarne. The C++ programming language / Bjarne Stroustrup. – Fourth edition. – М.: Pearson Education, Inc., 2003. – 1366 с.
13. Build a C# .NET app with WinUI 3 and Win32 interop // Microsoft [Электронный ресурс]. URL: <https://learn.microsoft.com/en-us/windows/apps/winui/winui3/desktop-winui3-app-with-basic-interop> (дата обращения: 24.05.2023).
14. Пахомов, Б. И. C# для начинающих. / Пахомов Б. И. – СПб.: БХВ-Петербург, 2014. — 432 с.: ил.
15. Kinney, Steven. Electron in Action / Steven Kinney – М.: Manning Publications Co., 2019. – 376 с.
16. Мазур, И. И. Управление проектами: Учебное пособие / Мазур И. И., Шапиро В. Д., Ольдерогге Н. Г. Под общ. ред. И. И. Мазура. – 2-е изд. — М.: Омега-Л, 2004. – с. 664.
17. Куликов, С.С. Тестирование программного обеспечения. Базовый курс (2-е издание) / Куликов С. С. – М.: EPAM Systems, 2017. – 312 с.
18. Copeland, Lee. A Practitioner's Guide to Software Test Design / Lee Copeland – М.: STQE Publishing, 2004. – 335 с.
19. Марков, А. С. Методы оценки несоответствия средств защиты информации / Марков А. С., Цирлов В. Л., Барабанов А. В. Под ред. А. С. Маркова. – М.: Радио и связь, 2012. – 192 с.
20. Laird, Linda M. Software measurement and estimation: a practical approach / Linda M. Laird, M. Carol Brennan. – М.: IEEE Computer Society., 2006.– 276 с.

Приложение А
ЛИСТИНГ КОДА ФАЙЛА WINDOWSDEVICEPROVIDER.H
(рекомендуемое)

```
#ifndef WINDOWSDEVICEPROVIDER_H
#define WINDOWSDEVICEPROVIDER_H

#include "abstractdeviceprovider.h"
#include <Windows.h>
#include <SetupAPI.h>
#include <memory>

class WindowsDeviceProvider : public AbstractDeviceProvider
{
    Q_OBJECT
public:
    explicit WindowsDeviceProvider(QObject *parent = nullptr);
public slots:
    void updateDevices() override;
};

#endif // WINDOWSDEVICEPROVIDER_H
```

Приложение Б

ЛИСТИНГ КОДА ФАЙЛА WINDOWSDEVICEPROVIDER.CPP

(рекомендуемое)

```

#include "windowsdeviceprovider.h"
#include "diskdevices.h"
#include "windowsdevice.h"

WindowsDeviceProvider::WindowsDeviceProvider(QObject *parent)
    : AbstractDeviceProvider{parent}
{

}

void WindowsDeviceProvider::updateDevices()
{
    m_devices.clear();

    DiskDevices diskDevices;

    for (DWORD device = 0; diskDevices.setDevice(device);
++device) {
        for (DWORD iface = 0; diskDevices.setInterface(iface);
++iface) {
            WindowsDevice *windowsDevice = new
WindowsDevice(this);
            windowsDevice->sethandle(diskDevices.handle());
            windowsDevice->setName(diskDevices.name());

m_devices.emplace_back(qobject_cast<AbstractDevice*>(windowsDevice
));
        }
    }

    emit devicesChanged();
}

```

Приложение В
ЛИСТИНГ КОДА ФАЙЛА WINDOWSDEVICE.H
(рекомендуемое)

```

#ifndef WINDOWSDEVICE_H
#define WINDOWSDEVICE_H

#include "abstractdevice.h"
#include "handle.h"

class WindowsDevice : public AbstractDevice
{
    Q_OBJECT
    Q_PROPERTY(Handle* handle READ handle WRITE sethandle NOTIFY
handleChanged)
public:
    explicit WindowsDevice(QObject *parent = nullptr);

    void prepare() override;
    quint64 writeBlock(void *data, qsize_t size) override;
    quint64 readBlock(void *data, qsize_t size) override;
    void setPositionRelative(qint64 pos) override;
    bool setPosition(qint64 pos) override;
    void resetState() override;

    quint64 size() const override;
    qsize_t sectorSize() const override;
    bool mount() const override;

    Handle *handle() const;
    void sethandle(Handle *newHandle);

signals:
    void deviceChanged();
    void handleChanged();
private:
    Handle *m_handle = nullptr;
    quint32 m_sectorSize;
};

#endif // WINDOWSDEVICE_H

```

Приложение Г
ЛИСТИНГ КОДА ФАЙЛА WINDOWSDEVICE.CPP
(рекомендуемое)

```

#include "windowsdevice.h"
#include "handle.h"
#include <QDebug>
#include "volumesearchhandle.h"
#include <QScopedPointer>

WindowsDevice::WindowsDevice(QObject *parent)
    : AbstractDevice{parent}
{

}

void WindowsDevice::prepare()
{
    handle()->create();
}

quint64 WindowsDevice::writeBlock(void *data, qsize_t size)
{
    if (size % m_sector_size) {
        qFatal() << "Size of block is not multiple of sector
size.";
        return 0;
    }

    return handle()->writeBlock(data, size);
}

quint64 WindowsDevice::readBlock(void *data, qsize_t size)
{
    if (size % m_sector_size) {
        qFatal() << "Size of block is not multiple of sector
size.";
        return 0;
    }

    return handle()->readBlock(data, size);
}

void WindowsDevice::setPositionRelative(qint64 pos)
{
    handle()->setPositionRelative(pos);
}

bool WindowsDevice::setPosition(qint64 pos)
{
    return handle()->setPointer(pos) != -1;
}

void WindowsDevice::resetState()
{
    handle()->setPointer(0);
}

```

```

quint64 WindowsDevice::size() const
{
    return handle()->size();
}

qsize_t WindowsDevice::sectorSize() const
{
    return m_sectorsize;
}

bool WindowsDevice::mount() const
{
    VolumeSearchHandle searchHandle;

    for (;;)
    {
        QScopedPointer<Handle> volume(searchHandle.next());

        if (volume->invalid())
            return false;
        if (volume->deviceNumber() == handle()->deviceNumber() &&
            volume->isMount())
            return true;
    }
}

Handle *WindowsDevice::handle() const
{
    return m_handle;
}

void WindowsDevice::sethandle(Handle *newHandle)
{
    if (m_handle == newHandle)
        return;
    m_handle = newHandle;
    m_handle->setParent(this);
    m_sectorsize = m_handle->sectorSize();
    emit handleChanged();
}

```

Приложение Д
ЛИСТИНГ КОДА ФАЙЛА DISKDEVICES.H
(рекомендуемое)

```

#ifndef DISKDEVICES_H
#define DISKDEVICES_H

#include <QObject>
#include <QString>
#include <Windows.h>
#include <SetupAPI.h>
#include "handle.h"

class DiskDevices : public QObject
{
    Q_OBJECT
public:
    explicit DiskDevices(QObject *parent = nullptr);
    ~DiskDevices();
    bool setDevice(DWORD index);
    bool setInterface(DWORD index);
    QString name();
    Handle *handle();
private:
    HDEVINFO diskClassDevices;
    GUID deviceClass = GUID_DEVINTERFACE_DISK;
    SP_DEVINFO_DATA deviceInfo;
    SP_DEVICE_INTERFACE_DATA deviceInterfaceData;
    SP_DRVINFO_DATA driverInfo;
    QScopedArrayPointer<char> buffer;
    DWORD requiredSize;

    void initDiskClassDevices();
    void createBuffer();
    PSP_DEVICE_INTERFACE_DETAIL_DATA deviceInterfaceDetailData();
    void setupDeviceInterfaceDetailData();
    void setName();
};

#endif // DISKDEVICES_H

```

Приложение Е
ЛИСТИНГ КОДА ФАЙЛА DISKDEVICES.CPP
(рекомендуемое)

```

#include "diskdevices.h"
#include "errorhandling.h"
#include <QScopedPointer>

DiskDevices::DiskDevices(QObject *parent)
    : QObject{parent}
{
    initDiskClassDevices();
    deviceInterfaceData.cbSize = sizeof(SP_DEVICE_INTERFACE_DATA);
    deviceInfo.cbSize = sizeof(SP_DEVINFO_DATA);
    driverInfo.cbSize = sizeof(SP_DRVINFO_DATA);
}

DiskDevices::~DiskDevices()
{
    if (!SetupDiDestroyDeviceInfoList(diskClassDevices))
        LogLastError();
}

void DiskDevices::initDiskClassDevices()
{
    diskClassDevices = SetupDiGetClassDevs(
        &deviceClass,
        NULL, NULL,
        DIGCF_PRESENT | DIGCF_DEVICEINTERFACE);

    if (diskClassDevices == INVALID_HANDLE_VALUE)
        LogLastError();
}

bool DiskDevices::setDevice(DWORD index)
{
    bool result = SetupDiEnumDeviceInfo(diskClassDevices, index,
    &deviceInfo);

    if (!result && GetLastError() != ERROR_NO_MORE_ITEMS)
        LogLastError();

    return result;
}

bool DiskDevices::setInterface(DWORD index)
{
    bool result = SetupDiEnumDeviceInterfaces(
        diskClassDevices,
        &deviceInfo,
        &deviceClass,
        index,
        &deviceInterfaceData);

    return result;
}

```

```

QString DiskDevices::name()
{
    DWORD requiredSize;

    if (!SetupDiGetDeviceRegistryProperty(
        diskClassDevices, &deviceInfo,
        SPDRP_FRIENDLYNAME, NULL,
        NULL, 0, &requiredSize) &&
        GetLastError() != ERROR_INSUFFICIENT_BUFFER)
    {
        LogLastError();
        return "";
    };

    QScopedArrayPointer<BYTE> buffer(new BYTE[requiredSize]);

    if (!SetupDiGetDeviceRegistryProperty(
        diskClassDevices, &deviceInfo,
        SPDRP_FRIENDLYNAME, NULL,
        buffer.get(), requiredSize, NULL))
    {
        LogLastError();
        return "";
    };

    return
    QString::fromWCharArray(reinterpret_cast<LPWSTR>(buffer.get()),
        requiredSize / sizeof(WCHAR));
}

Handle *DiskDevices::handle()
{
    setupDeviceInterfaceDetailData();
    Handle *handle = new Handle(this);
    handle->openByName(deviceInterfaceDetailData()->DevicePath);
    return handle;
}

void DiskDevices::createBuffer()
{
    if (SetupDiGetDeviceInterfaceDetail(
        diskClassDevices,
        &deviceInterfaceData,
        NULL, 0, &requiredSize, NULL) ||
        GetLastError() != ERROR_INSUFFICIENT_BUFFER)
    {
        LogLastError();
        return;
    }

    buffer.reset(new char[requiredSize]);
    deviceInterfaceDetailData()->cbSize =
    sizeof(SP_DEVICE_INTERFACE_DETAIL_DATA);
}

```

```

}

PSP_DEVICE_INTERFACE_DETAIL_DATA
DiskDevices::deviceInterfaceDetailData()
{
    return
reinterpret_cast<PSP_DEVICE_INTERFACE_DETAIL_DATA>(buffer.data());
}

void DiskDevices::setupDeviceInterfaceDetailData()
{
    createBuffer();

    if (!SetupDiGetDeviceInterfaceDetail(
        diskClassDevices,
        &deviceInterfaceData,
        deviceInterfaceDetailData(),
        requiredSize, NULL, NULL))
    {
        LogLastError();
    }
}

```

Приложение Ж
ЛИСТИНГ КОДА ФАЙЛА HANDLE.H
(рекомендуемое)

```

#ifndef HANDLE_H
#define HANDLE_H

#include <QObject>
#include <QString>
#include <Windows.h>

class Handle : public QObject
{
    Q_OBJECT
    Q_PROPERTY(HANDLE handle READ handle WRITE sethandle NOTIFY
handleChanged)
    Q_PROPERTY(quint64 size READ size NOTIFY sizeChanged)
public:
    explicit Handle(QObject *parent = nullptr);
    ~Handle();

    HANDLE handle() const;
    void sethandle(HANDLE newHandle);

    void openByName(LPWSTR name);
    quint32 writeBlock(void *data, qsize_t size);
    quint32 readBlock(void *data, qsize_t size);
    quint64 setPointer(quint64 pointer);
    quint64 setPositionRelative(quint64 position);

    quint64 size();
    int deviceNumber();
    qsize_t sectorSize();
    bool isMount();
    bool invalid();

    void dismount();
    void offline();
    void lock();
    void unlock();
    void create();
    quint64 diskSize();
signals:
    void handleChanged();
    void sizeChanged();
    void errorMessageChanged();

private:
    HANDLE m_handle;
};

#endif // HANDLE_H

```

Приложение И
ЛИСТИНГ КОДА ФАЙЛА HANDLE.CPP
(рекомендуемое)

```

#include "handle.h"
#include "errorhandling.h"
#include <ostream>
#include <winnt.h>
#include <QDebug>

Handle::Handle(QObject *parent)
    : QObject{parent}, m_handle(INVALID_HANDLE_VALUE)
{

}

Handle::~~Handle()
{
    if (!invalid() && !CloseHandle(handle()))
        LogLastError();
}

HANDLE Handle::handle() const
{
    return m_handle;
}

void Handle::sethandle(HANDLE newHandle)
{
    if (m_handle == newHandle)
        return;
    m_handle = newHandle;
    emit handleChanged();
}

void Handle::openByName(LPWSTR name)
{
    sethandle(CreateFile(
        name,
        GENERIC_READ | GENERIC_WRITE,
        FILE_SHARE_READ | FILE_SHARE_WRITE,
        NULL,
        OPEN_EXISTING,
        FILE_FLAG_NO_BUFFERING,
        NULL));

    if (handle() == INVALID_HANDLE_VALUE)
        LogLastError();
}

quint32 Handle::writeBlock(void *data, qsize_t size)
{
    DWORD numberOfBytesWritten = 0;

    if (!WriteFile(
        handle(),
        data,

```

```

        size,
        &numberOfBytesWritten,
        NULL))
    {
        LogLastError();
    }

    return numberOfBytesWritten;
}

qint32 Handle::readBlock(void *data, qsize_t size)
{
    DWORD numberOfBytesRead = 0;

    if (!ReadFile(
        handle(),
        data,
        size,
        &numberOfBytesRead,
        NULL))
    {
        LogLastError();
    }

    return numberOfBytesRead;
}

qint64 Handle::setPointer(qint64 pointer)
{
    LARGE_INTEGER pointerUnion;
    pointerUnion.QuadPart = pointer;

    pointerUnion.LowPart = SetFilePointer(
        handle(),
        pointerUnion.LowPart,
        &pointerUnion.HighPart,
        FILE_BEGIN);

    if (pointerUnion.LowPart == INVALID_SET_FILE_POINTER)
    {
        DWORD error = GetLastError();

        if (error != NO_ERROR) {
            pointerUnion.QuadPart = -1;
            LogError(error);
        }
    }

    return pointerUnion.QuadPart;
}

qint64 Handle::setPositionRelative(qint64 position)
{

```

```

LARGE_INTEGER pointerUnion;
pointerUnion.QuadPart = position;

pointerUnion.LowPart = SetFilePointer(
    handle(),
    pointerUnion.LowPart,
    &pointerUnion.HighPart,
    FILE_CURRENT);

if (pointerUnion.LowPart == INVALID_SET_FILE_POINTER)
{
    DWORD error = GetLastError();

    if (error != NO_ERROR) {
        pointerUnion.QuadPart = -1;
        LogError(error);
    }
}

return pointerUnion.QuadPart;
}

int Handle::deviceNumber()
{
    STORAGE_DEVICE_NUMBER diskNumber;

    if (!DeviceIoControl(
        handle(),
        IOCTL_STORAGE_GET_DEVICE_NUMBER,
        NULL,
        0,
        &diskNumber,
        sizeof(STORAGE_DEVICE_NUMBER),
        NULL,
        NULL))
    {
        LogLastError();
    }

    return diskNumber.DeviceNumber;
}

bool Handle::isMount()
{
    return DeviceIoControl(
        handle(), FSCTL_IS_VOLUME_MOUNTED,
        NULL, 0, NULL, 0, NULL, NULL);
}

bool Handle::invalid()
{
    return handle() == INVALID_HANDLE_VALUE;
}

```

```

void Handle::dismount()
{
    if (!DeviceIoControl(
        handle(),
        FSCTL_DISMOUNT_VOLUME,
        NULL,
        0,
        NULL,
        0,
        NULL,
        NULL))
    {
        LogLastError();
    }
}

void Handle::offline()
{
    if (!DeviceIoControl(
        handle(),
        IOCTL_VOLUME_OFFLINE,
        NULL,
        0,
        NULL,
        0,
        NULL,
        NULL))
    {
        LogLastError();
    }
}

void Handle::lock()
{
    if (!DeviceIoControl(
        handle(),
        FSCTL_LOCK_VOLUME,
        NULL,
        0,
        NULL,
        0,
        NULL,
        NULL))
    {
        LogLastError();
    }
}

void Handle::unlock()
{
    if (!DeviceIoControl(
        handle(),

```

```

        FSCTL_UNLOCK_VOLUME,
        NULL,
        0,
        NULL,
        0,
        NULL,
        NULL) )
    {
        LogLastError();
    }
}

void Handle::create()
{
    CREATE_DISK disk;
    disk.PartitionStyle = PARTITION_STYLE_GPT;

    if (!DeviceIoControl(
        handle(),
        IOCTL_DISK_CREATE_DISK,
        &disk,
        sizeof(CREATE_DISK),
        NULL,
        0,
        NULL,
        NULL) )
    {
        LogLastError();
    }
}

qsize_t Handle::sectorSize()
{
    DISK_GEOMETRY_EX geometry;

    if (!DeviceIoControl(
        handle(),
        IOCTL_DISK_GET_DRIVE_GEOMETRY_EX,
        NULL,
        0,
        &geometry,
        sizeof(DISK_GEOMETRY_EX),
        NULL,
        NULL) )
    {
        LogLastError();
    }

    return geometry.Geometry.BytesPerSector;
}

```

Приложение К
ЛИСТИНГ КОДА ФАЙЛА VOLUMESEARCHHANDLE.H
(рекомендуемое)

```

#ifndef VOLUMESEARCHHANDLE_H
#define VOLUMESEARCHHANDLE_H

#include <QObject>
#include <Windows.h>
#include "handle.h"

class VolumeSearchHandle : public QObject
{
    Q_OBJECT
public:
    explicit VolumeSearchHandle(QObject *parent = nullptr);
    ~VolumeSearchHandle();
    Handle *next();
private:
    HANDLE handle = INVALID_HANDLE_VALUE;
    bool iterate(LPWSTR path, DWORD size);
    bool checkHandle();
};

#endif // VOLUMESEARCHHANDLE_H

```

Приложение Л
ЛИСТИНГ КОДА ФАЙЛА VOLUMESEARCHHANDLE.CPP
(рекомендуемое)

```

#include "volumesearchhandle.h"
#include "errorhandling.h"
#include <fileapi.h>
#include <string.h>
VolumeSearchHandle::VolumeSearchHandle(QObject *parent)
    : QObject{parent}
{
}

VolumeSearchHandle::~VolumeSearchHandle()
{
    if (!FindVolumeClose(handle))
        LogLastError();
}

Handle *VolumeSearchHandle::next()
{
    WCHAR path[256];
    Handle *handle = new Handle(this);
    if (!iterate(path, sizeof(path)))
        return handle;
    path[wcslen(path) - 1] = L'\0';
    handle->openByName(path);
    return handle;
}

bool VolumeSearchHandle::iterate(LPWSTR path, DWORD size)
{
    if (handle == INVALID_HANDLE_VALUE) {
        handle = FindFirstVolume(path, size);
        return checkHandle();
    }
    else if (!FindNextVolume(handle, path, size)) {
        if (GetLastError() != ERROR_NO_MORE_FILES)
            LogLastError();
        return false;
    }

    return true;
}

bool VolumeSearchHandle::checkHandle()
{
    if (handle == INVALID_HANDLE_VALUE) {
        LogLastError();
        return false;
    }

    return true;
}

```

Приложение М
ЛИСТИНГ КОДА ФАЙЛА TESTWORKER.H
(рекомендуемое)

```

#ifndef TESTWORKER_H
#define TESTWORKER_H

#include <QObject>
#include <QScopedArrayPointer>
#include <functional>
#include "abstractdataprovder.h"
#include "abstractdevice.h"

class TestWorker : public QObject
{
    Q_OBJECT
    Q_PROPERTY(AbstractDataProvider* dataProvider READ
dataProvider WRITE setDataProvider NOTIFY dataProviderChanged)
    Q_PROPERTY(AbstractDevice* device READ device WRITE setDevice
NOTIFY deviceChanged)
    Q_PROPERTY(qsizetype sectorSize READ sectorSize WRITE
setsectorSize NOTIFY sectorSizeChanged)
    Q_PROPERTY(int percent READ percent WRITE setpercent NOTIFY
percentChanged)
public:
    explicit TestWorker(QObject *parent = nullptr);
    void test();

    AbstractDataProvider *dataProvider() const;
    AbstractDevice *device() const;
    qsizetype sectorSize() const;

    int percent() const;
    void setpercent(int newPercent);

public slots:
    void setDataProvider(AbstractDataProvider *newDataProvider);
    void setDevice(AbstractDevice *newDevice);
    void setsectorSize(qsizetype newSectorSize);

private slots:
    void reallocateBuffer();

signals:
    void dataProviderChanged();
    void deviceChanged();
    void sectorSizeChanged();
    void badBlockFound(quint64);
    void progressChanged(quint64);
    void writeSpeedChanged(quint64);
    void readSpeedChanged(quint64);

    void percentChanged();

private:

```

```

static constexpr qsize_t BLOCK_COUNT = 262144;

void writeBlock();
bool writeChunk();
bool readChunk(qsize_t iteration);
void compareChunk(qsize_t iteration, qsize_t written);
quint64 measureSpeed(std::function<qsize_t()> f);

AbstractDataProvider *m_dataProvider = nullptr;
AbstractDevice *m_device = nullptr;
qsize_t m_sectorSize;
QScopedArrayPointer<char> deviceBuffer;
QScopedArrayPointer<char> compareBuffer;
int m_percent;
};

#endif // TESTWORKER_H

```

Приложение Н
ЛИСТИНГ КОДА ФАЙЛА TESTWORKER.CPP
(рекомендуемое)

```

#include "testworker.h"
#include <chrono>
#include <QThread>
#include <algorithm>

TestWorker::TestWorker(QObject *parent)
    : QObject{parent}
{
    connect(this, &TestWorker::sectorSizeChanged,
&TestWorker::reallocateBuffer);
    connect(this, &TestWorker::deviceChanged, this, [this]() {
setsectorSize( device()->sectorSize() ); });
}

void TestWorker::test()
{
    device()->setPosition(device()->size() - device()-
>sectorSize());

    for (qsize_t iteration = 0;; iteration += 100)
    {
        for (qsize_t i = 0; i < percent(); ++i) {
            if (QThread::currentThread()-
>isInterruptionRequested() ||
                (writeChunk() && readChunk(i)))
            {
                return;
            }

            bool stop = writeChunk();
            stop |= readChunk(iteration);

            if (stop)
                return;
        }

        quint64 progress = (iteration + 100) * BLOCK_COUNT *
device()->sectorSize();
        emit progressChanged(progress);

        if (!device()->setPosition(progress))
        {
            return;
        }
    }
}

bool TestWorker::writeChunk()
{
    bool stop = false;
    dataProvider()->getDataBlock(deviceBuffer.data(), BLOCK_COUNT
* sectorSize());

```

```

        std::copy(deviceBuffer.data(), deviceBuffer.data() +
BLOCK_COUNT * sectorSize(), compareBuffer.data());

        quint64 speed = measureSpeed([this, &stop]() {
            for (qsize_t i = 0; i < BLOCK_COUNT; ++i) {
                qsize_t written = device()-
>writeBlock(deviceBuffer.data() + i * sectorSize(), sectorSize());

                if (written == 0) {
                    stop = true;
                    return i;
                }
            }

            return BLOCK_COUNT;
        });

        emit writeSpeedChanged(speed);
        return stop;
    }

bool TestWorker::readChunk(qsize_t iteration)
{
    bool stop = false;
    device()->setPositionRelative(-1 * (BLOCK_COUNT *
sectorSize()));

    quint64 speed = measureSpeed([this, &stop, iteration]() {
        for (qsize_t i = 0; i < BLOCK_COUNT; ++i) {
            auto read = device()->readBlock(deviceBuffer.data() +
i * sectorSize(), sectorSize());

            if (read == 0) {
                stop = true;
                compareChunk(iteration, i);
                return i;
            }
        }

        compareChunk(iteration, BLOCK_COUNT);
        return BLOCK_COUNT;
    });

    emit readSpeedChanged(speed);
    return stop;
}

void TestWorker::compareChunk(qsize_t iteration, qsize_t
written)
{
    for (qsize_t i = 0; i < written; ++i) {
        if (!std::equal(
            deviceBuffer.data() + i * sectorSize(),

```

```

        deviceBuffer.data() + (i + 1) * sectorSize(),
        compareBuffer.data() + i * sectorSize()))
    {
        emit badBlockFound((iteration * BLOCK_COUNT + i) *
device()->sectorSize());
    }
}

void TestWorker::writeBlock()
{
    dataProvider()->getDataBlock(deviceBuffer.data(),
sectorSize());
    device()->writeBlock(deviceBuffer.data(), sectorSize());
}

quint64 TestWorker::measureSpeed(std::function<qsize_t> f)
{
    auto start = std::chrono::high_resolution_clock::now();
    auto written = f();
    auto time = std::chrono::high_resolution_clock::now() - start;
    auto ns =
std::chrono::duration_cast<std::chrono::nanoseconds>(time);
    quint64 speed = sectorSize() * written * 1e9 / ns.count();
    return speed;
}

AbstractDataProvider *TestWorker::dataProvider() const
{
    return m_dataProvider;
}

void TestWorker::setDataProvider(AbstractDataProvider
*newDataProvider)
{
    if (m_dataProvider == newDataProvider)
        return;
    m_dataProvider = newDataProvider;
    emit dataProviderChanged();
}

AbstractDevice *TestWorker::device() const
{
    return m_device;
}

void TestWorker::setDevice(AbstractDevice *newDevice)
{
    if (m_device == newDevice)
        return;
    m_device = newDevice;
    emit deviceChanged();
}

```

```

qsize_t TestWorker::sectorSize() const
{
    return m_sectorSize;
}

void TestWorker::setsectorSize(qsize_t newSectorSize)
{
    if (m_sectorSize == newSectorSize)
        return;
    m_sectorSize = newSectorSize;
    emit sectorSizeChanged();
}

void TestWorker::reallocateBuffer()
{
    deviceBuffer.reset(new char[sectorSize() * BLOCK_COUNT]);
    compareBuffer.reset(new char[sectorSize() * BLOCK_COUNT]);
}

int TestWorker::percent() const
{
    return m_percent;
}

void TestWorker::setpercent(int newPercent)
{
    if (m_percent == newPercent)
        return;
    m_percent = newPercent;
    emit percentChanged();
}

```

Приложение П
ЛИСТИНГ КОДА ФАЙЛА TESTSERVICE.H
(рекомендуемое)

```

#ifndef TESTSERVICE_H
#define TESTSERVICE_H

#include <QObject>
#include <QThread>
#include "abstractdataproducer.h"
#include "abstractdevice.h"
#include "testworker.h"

class TestService : public QObject
{
    Q_OBJECT
    Q_PROPERTY(AbstractDataProvider* dataProvider READ
dataProvider WRITE setDataProvider NOTIFY dataProviderChanged)
    Q_PROPERTY(AbstractDevice* device READ device WRITE setDevice
NOTIFY deviceChanged)
    Q_PROPERTY(qsizetype sectorSize READ sectorSize WRITE
setsectorSize NOTIFY sectorSizeChanged)
    Q_PROPERTY(int percent READ percent WRITE setpercent NOTIFY
percentChanged)
public:
    explicit TestService(QObject *parent = nullptr);
    AbstractDataProvider *dataProvider() const;
    AbstractDevice *device() const;
    qsizetype sectorSize() const;
    int percent() const;
    void setpercent(int newPercent);
public slots:
    void setDataProvider(AbstractDataProvider *newDataProvider);
    void setDevice(AbstractDevice *newDevice);
    void setsectorSize(qsizetype newSectorSize);
    void stop();
signals:
    void dataProviderChanged(AbstractDataProvider*);
    void deviceChanged(AbstractDevice*);
    void sectorSizeChanged(qsizetype);
    void badBlockFound(quint64);
    void progressChanged(quint64);
    void writeSpeedChanged(quint64);
    void readSpeedChanged(quint64);
    void test();
    void percentChanged(int);
private:
    QThread *thread = nullptr;
    TestWorker *worker = nullptr;
    int m_percent;
};

#endif // TESTSERVICE_H

```

Приложение Р
ЛИСТИНГ КОДА ФАЙЛА TESTSERVICE.CPP
(рекомендуемое)

```

#include "testservice.h"
#include "testworker.h"

TestService::TestService(QObject *parent)
    : QObject{parent}
{
    worker = new TestWorker();
    thread = new QThread(this);
    worker->moveToThread(thread);

    connect(this, &TestService::dataProviderChanged, worker,
&TestWorker::setdataProvider);
    connect(this, &TestService::deviceChanged, worker,
&TestWorker::setDevice);
    connect(this, &TestService::sectorSizeChanged, worker,
&TestWorker::setsectorSize);
    connect(this, &TestService::test, worker, &TestWorker::test);
    connect(this, &TestService::percentChanged, worker,
&TestWorker::setpercent);
    connect(worker, &TestWorker::badBlockFound, this,
&TestService::badBlockFound);
    connect(worker, &TestWorker::progressChanged, this,
&TestService::progressChanged);
    connect(worker, &TestWorker::readSpeedChanged, this,
&TestService::readSpeedChanged);
    connect(worker, &TestWorker::writeSpeedChanged, this,
&TestService::writeSpeedChanged);

    thread->start();
}

AbstractDataProvider *TestService::dataProvider() const
{
    return worker->dataProvider();
}

void TestService::setdataProvider(AbstractDataProvider
*newDataProvider)
{
    emit dataProviderChanged(newDataProvider);
}

AbstractDevice *TestService::device() const
{
    return worker->device();
}

void TestService::setDevice(AbstractDevice *newDevice)
{
    emit deviceChanged(newDevice);
}

qsize_t TestService::sectorSize() const

```

```
{
    return worker->sectorSize();
}

void TestService::stop()
{
    thread->requestInterruption();
}

void TestService::setsectorSize(qsizetype newSectorSize)
{
    emit sectorSizeChanged(newSectorSize);
}

int TestService::percent() const
{
    return worker->percent();
}

void TestService::setpercent(int newPercent)
{
    emit percentChanged(newPercent);
}
```

Приложение С
РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ
(рекомендуемое)

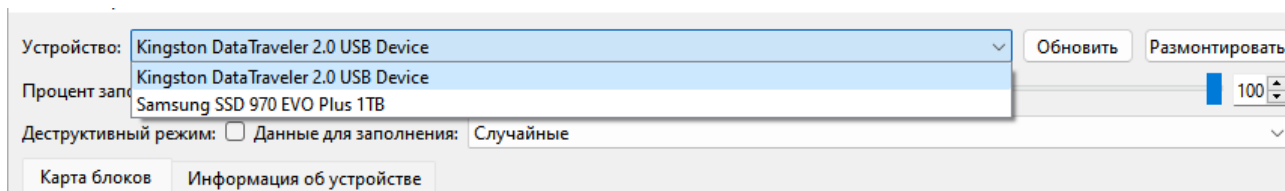


Рисунок С.1 – Тест выбора накопителя

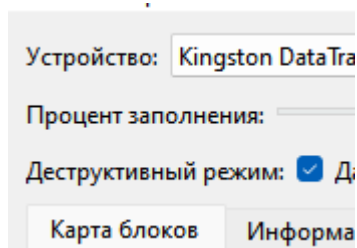


Рисунок С.2 – Выбор деструктивного/недеструктивного режима

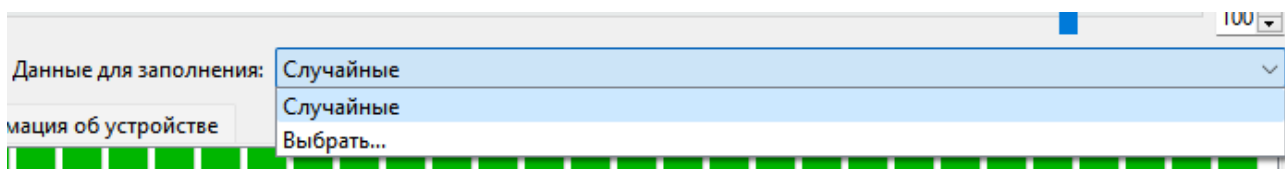


Рисунок С.3 – Выбор данных для заполнения

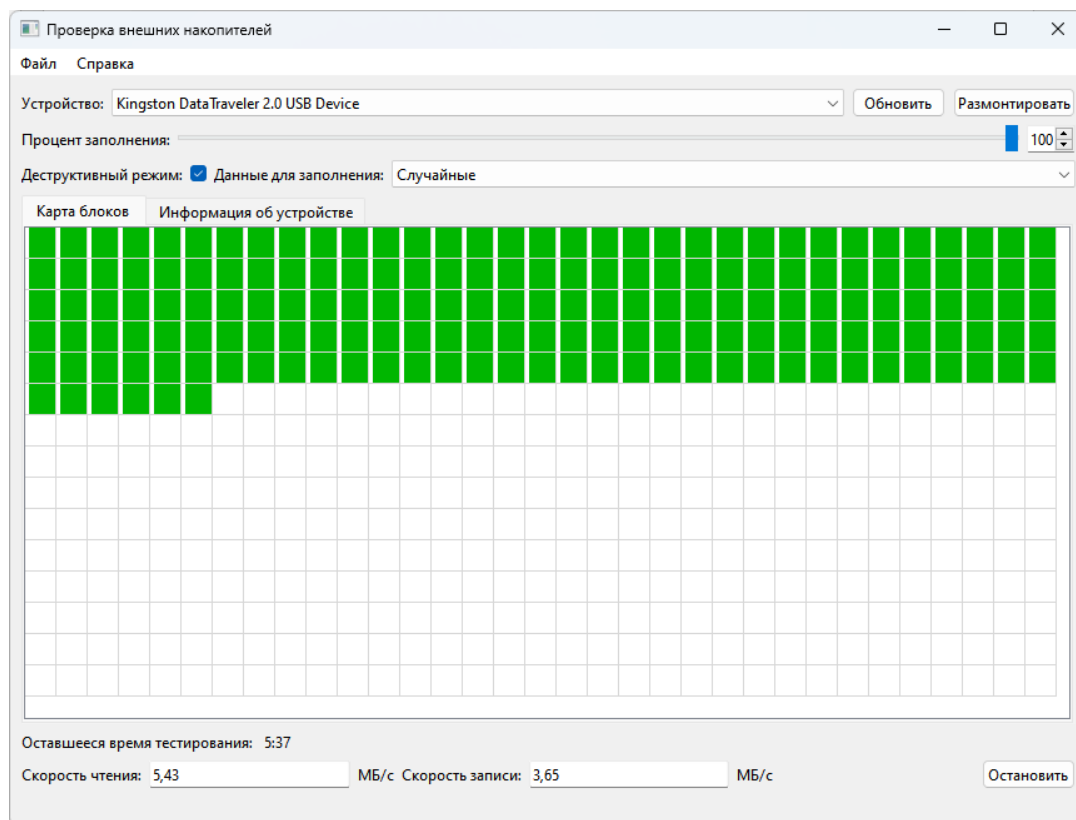


Рисунок С.4 – Процесс тестирования накопителя

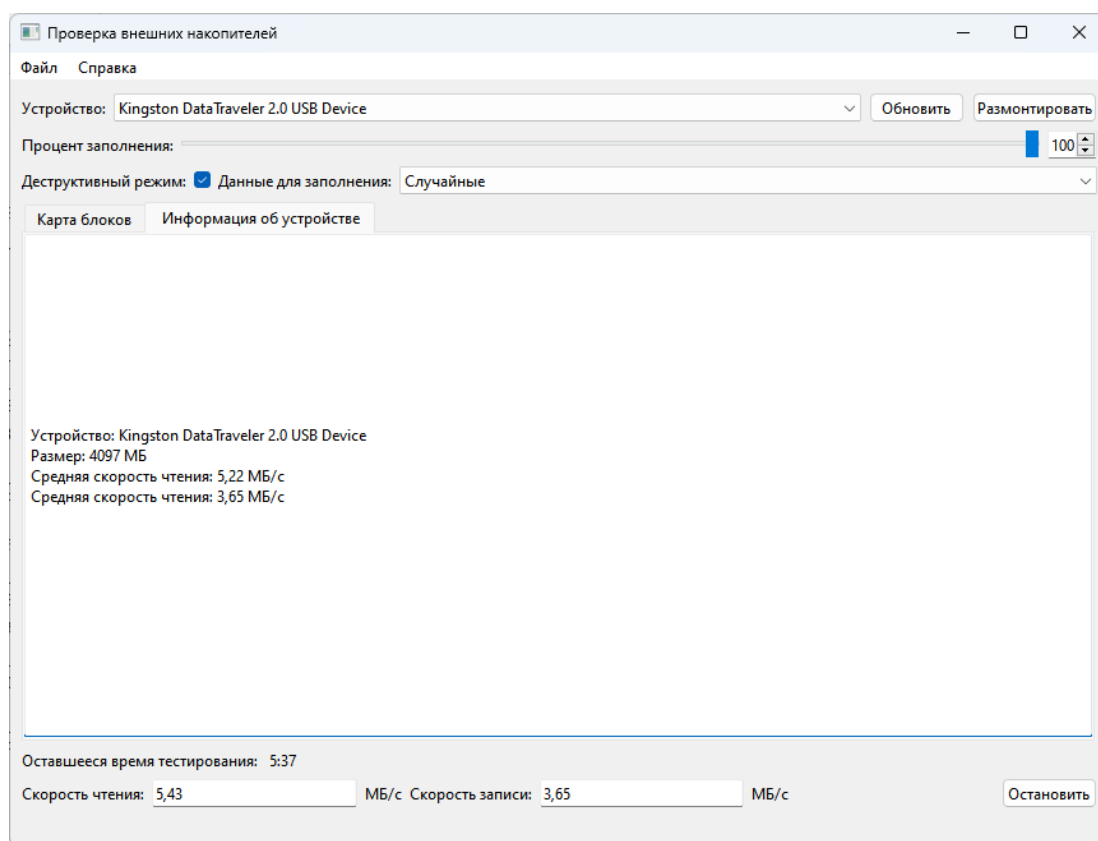


Рисунок С.5 – Информация об устройстве

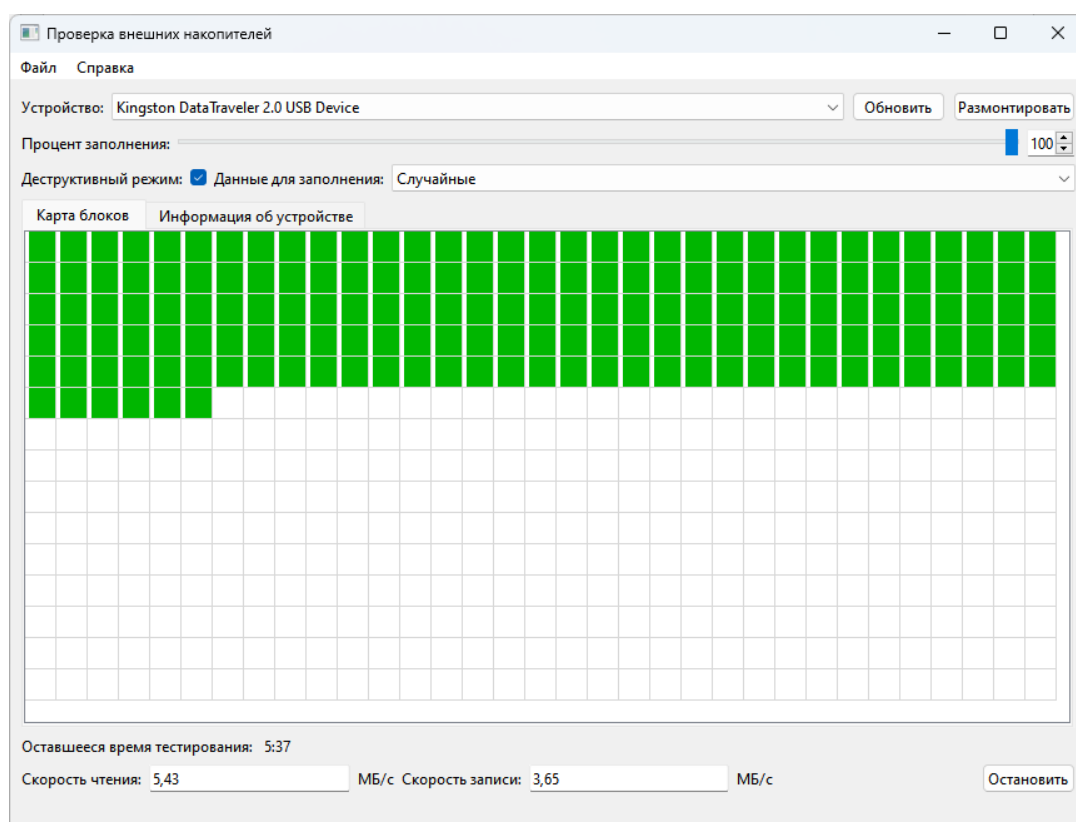
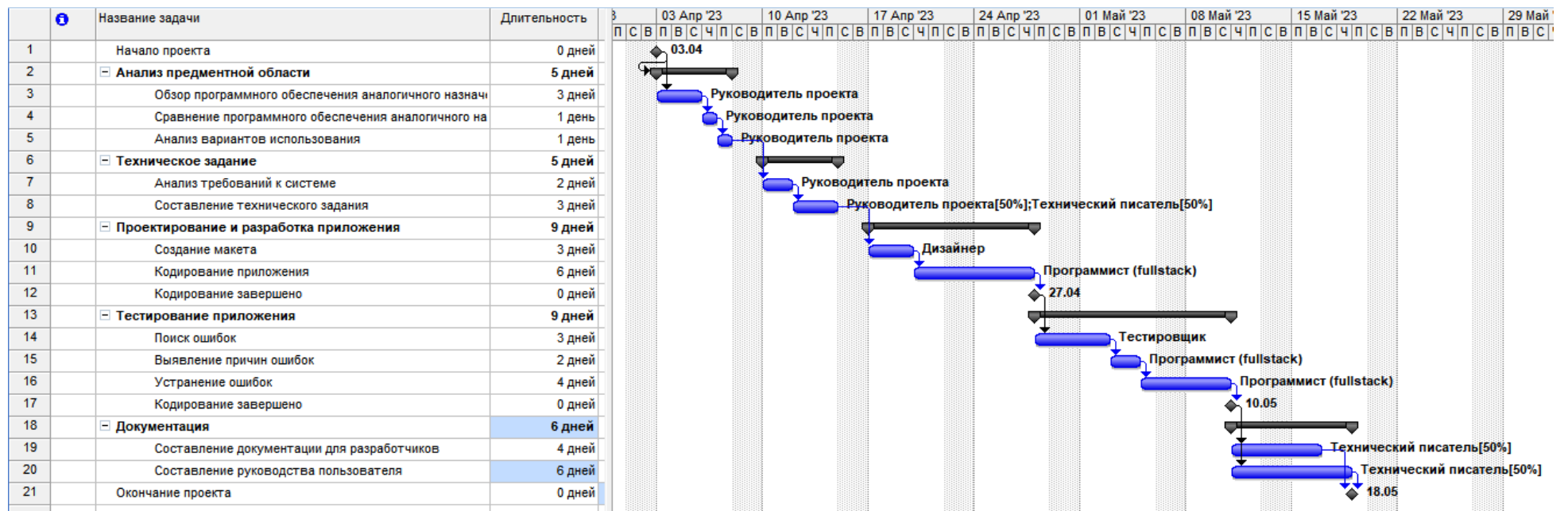


Рисунок С.6 – Просмотр оставшегося времени проверки

Приложение Т
ДИАГРАММА ГАНТА
(рекомендуемое)



Приложение У
ГРАФ ПОТОКА УПРАВЛЕНИЯ
(рекомендуемое)

