МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

КАФЕДРА КОМП'ЮТЕРНИХ НАУК

КВАЛІФІКАЦІЙНА МАГІСТЕРСЬКА РОБОТА

на тему:

«Інформаційна система контролю, аналізу та управління розміткою жорсткого диска»

Завідувач випускаючої кафедри — **Довбиш А.С.**

Керівник роботи — **Берест О.Б.**

Студента групи ІН.мз-91с — **Бойко С.О.**

СУМИ 2020

Сумський державний університет

(назва вузу)

Факультет *ЦЗДВФН* Кафедра *Комп'ютерних наук*

Спеціальність *«Інформатика»*

Затверджую:
зав.кафедрою _____
_____
"_____" _____ 20___р.

ЗАВДАННЯ
НА ДИПЛОМНИЙ ПРОЕКТ (РОБОТУ) СТУДЕНТОВІ

Бойко Сергій Олександрович

(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) *Інформаційна система контролю, аналізу та управління розміткою жорсткого диска*

затверджую наказом по інституту від "____" _____ 20 ___ р. № _____

2. Термін здачі студентом закінченого проекту (роботи) _____

3. Вхідні данні до проекту (роботи) _____
_____
_____
_____

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)
1) Аналітичний огляд літератури по темі випускної роботи та аналіз проблеми. Постановка задачі дослідження.
2) Розроблення інформаційної моделі контролю, аналізу та управління розміткою жорсткого диска.
3) Розробка інформаційного та програмного забезпечення системи.
4) Висновки та пропозиції щодо вдосконалення системи оцінювання.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) _____
_____
_____
_____
_____

6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання _____

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів дипломного проекту (роботи)	Термін виконання проекту (роботи)	Примітка
1.	*Аналітичний огляд літератури по темі випускної роботи та аналіз проблеми. Постановка задачі дослідження*	*20.09.2020*	
2.	*Розроблення інформаційної моделі контролю, аналізу та управління розміткою жорсткого диска*	*29.10.2020*	
3.	*Розробка інформаційного та програмного забезпечення системи*	*20.11.2020*	
4.	*Висновки та пропозиції щодо вдосконалення системи оцінювання*	*26.11.2020*	

Студент – дипломник _____
(підпис)

Керівник проекту _____
(підпис)

Зміст

Перелік скорочень

MBR - (Master Boot Record) головний завантажувальний запис, необхідна для подальшого завантаження операційної системи.

GPT - (GUID Partition Table) стандарт формату розміщення таблиць розділів на фізичному жорсткому диску.

FAT - (File Allocation Table) класична архітектура файлової системи.

IDE - (Integrated Development Environment) інтегроване середовище розробки.

MVC - (Model View Controller) модель уявлення контролер. ОС - операційна система.

NDIS - (Network Driver Interface Specification) - специфікація інтерфейсу мережевого драйвера.

ООП - об'єктно-орієнтоване програмування;

ЖД - жорсткий диск.

ПО - програмне забезпечення.

Вступ

Розроблена програма являє собою додаток для контролю, управління і аналізу розмітки жорсткого диска. Додаток дозволяє виконувати основні операції над файловою системою, такі як форматування, створення і видалення розділу, зміна типу завантажувального запису і т.д.

Основною причиною, що стала поштовхом для написання подібної програми, є проблема взаємодії користувача з файловою системою в операційній системі. В цей час існує ряд файлових систем, кожна з яких має свої переваги й недоліки і призначена для виконання завдань різного роду. Дуже часто у користувача можуть виникати завдання зміни розмітки жорсткого диска. Користувачеві може знадобитися відформатувати розділ, щоб була можливість працювати з іншою файловою системою. Може виникнути завдання створення нового розділу певного розміру. Так само з багатьох причин може виникнути необхідність зміни мітки активності розділу або типу завантажувального запису.

Таким чином, розробка подібного програмного продукту є актуальним завданням, здатної підвищити зручність роботи з файловими системами.

1 Аналіз предметної області та функціональних вимог до проєктної задачі

Завданням дипломного проєктування є написання і налагодження додатка для управління розміткою жорсткого диска development серверу на проєкті “O2”, що дозволить виконувати основні операції над дисками і розділами.

Додаток має забезпечувати можливість створювати і видаляти розділи жорсткого диска, форматувати їх в одну з доступних ОС файлової системи, змінювати тип завантажувального запису диска, змінювати букву або мітку тому, а також представляти актуальну інформацію про стан файлової системи.

У додатку повинна бути форма властивостей розділу, що відображає параметри розділу, такі як буква, мітка, розмір кластера і т.д.

Необхідною вимогою вважається наявність графічного призначеного для користувача інтерфейсу.

Додаток має використовувати бібліотеки операційної системи по роботі з розміткою жорсткого диска.

На Рис 1 зображена діаграма варіантів використання розробленого додатка. У таблицях 1, 2 відображені специфікації прецедентів створення ЖД і видалення ЖД.

Рисунок 1 - Діаграма варіантів використання

Таблиця 1 - Специфікація прецеденту «Створити розділ ЖД»

Прецедент: Створити розділ ЖД.
Id: 1
Короткий опис: Створення розділу ЖД з обраного вільного простору на диску.
Головні актори: Користувач.
Другорядні актори: Ні.
Передумови: Прецедент починається, коли користувач вибрав пункт меню «Створити розділ».

Продовження таблиці 1

Основний потік: 1. Додаток відкриває форму створення розділу 2. Користувач вибирає необхідні параметри створення розділу 3. Програма викликає функцію створення розділу з некерованої бібліотеки, закриває форму і оновлює список розділів і дисків
Післяумови: 1. Програма повертається в режим очікування команд користувача.
Альтернативні потоки: Ні.

Таблиця 2 - Технічна специфікація прецеденту «Видалити розділ ЖД»

Прецедент: Видалити розділ ЖД.
Id: 2
Короткий опис: Видалення обраного розділу ЖД.
Головні актори: Користувач.
Другорядні актори: Ні.
Передумови: Прецедент починається, коли користувач вибрав пункт меню «Видалити».
Основний потік: 1. Додаток відкриває форму підтвердження видалення 2. Програма викликає функцію видалення розділу з некерованої бібліотеки, закриває форму і оновлює список розділів і дисків
Післяумови: 1. Програма повертається в режим очікування команд користувача.
Альтернативні потоки: Ні.

Першим варіантом використання програми є створення розділу жорсткого диска з незайнятого простору або розширеного розділу. Для цього користувачеві необхідно вибрати відповідний пункт меню і вказати параметри створення розділу.

Другим варіантом використання є видалення розділу жорсткого диска. Для здійснення цієї дії користувачеві необхідно виділити в списку потрібний розділ і вибрати пункт меню «видалити розділ». При цьому додаток попросить підтвердження дії, оскільки при видаленні розділу жорсткого диска можлива втрата даних. У таблицях 3, 4 відображені специфікації прецедентів форматування розділу і зміна активності розділу.

Таблиця 3 - Специфікація прецеденту «Форматувати розділ ЖД»

Прецедент: Форматувати розділ ЖД.
Id: 3
Короткий опис: Форматування вибраного розділу ЖД.
Головні актори: Користувач.
Другорядні актори: Ні.
Передумови: Прецедент починається, коли користувач вибрав пункт меню «Форматувати».
Основний потік: 1. Додаток відкриває форму форматування 2. Користувач вибирає параметри форматування і натискає кнопку «Форматувати». 3. Програма створює окрему нитку, переводить інтерфейс форми в режим очікування.
Післяумови: 1. Програма повертається в режим очікування команд користувача.
Альтернативні потоки: В окремому потоці викликається функція форматування розділу, яка при завершенні закриває форму форматування і повертає програму в режим очікування команд користувача.

Таблиця 4 - Специфікація прецеденту «Змінити активність розділу»

Прецедент: Змінити активність розділу.
Id: 4
Короткий опис: Зміна прапора активності, обрана Вами ЖД.
Головні актори: Користувач.
Другорядні актори: Ні.
Передумови: Прецедент починається, коли користувач вибрав пункт меню «Зробити активним» або «Прибрати активність».
Основний потік: 1. Програма викликає функцію зміни мітки активності з некерованої бібліотеки і оновлює список розділів і дисків.
Післяумови: 1. Програма повертається в режим очікування команд користувача.
Альтернативні потоки: Ні.

Третім варіантом використання є форматування розділу жорсткого диска. Користувач виділяє розділ в списку і вибирає пункт меню «Форматувати розділ». У формі, що з'явилася користувач задає необхідні параметри, такі як файлова система, в яку форматується розділ, і то, варто виконати очищення змісту або ж повністю очистити розділ.

Додаток дозволяє змінювати активність одного з розділів вибором відповідного пункту меню. При цьому мітка активності знімається з поточного активного розділу. Мітка активності служить для визначення тому з операційною системою при включенні комп'ютера. Специфікація прецеденту зміни активності розділу відображена в таблиці 5.

Таблиця 5 - Технічна специфікація прецеденту «Змінити тип завантажувального запису»

Прецедент: Змінити тип завантажувального запису.
Id: 5
Короткий опис: Зміна типу завантажувального запису ЖД.
Головні актори: Користувач.
Другорядні актори: Ні.
Передумови: Прецедент починається, коли користувач вибрав пункт меню «Перетворити в MBR» або «Перетворити в GPT».
Основний потік: **1.** Додаток відкриває форму підтвердження зміни типу заголовної таблиці ЖД. **2.** Програма викликає функцію зміни заголовної таблиці ЖД з некерованої бібліотеки, закриває форму і оновлює список розділів і дисків
Післяумови: 3. Програма повертається в режим очікування команд користувача.
Альтернативні потоки: Ні.

Як випливає і таблиць прецедентів, все варіанти використання програми є досить типовими. Спочатку користувачем вибирається необхідна дія над диском і розділом, потім додаток запитує підтвердження цієї дії, а також пропонує користувачеві встановити необхідні параметри. Потім додаток виконує цю дію і повертається в режим очікування команд користувача.

Користувач також може змінити тип завантажувального запису диска, тобто сформувати таблицю розділів у вигляді MBR або GPT. Для цього користувач повинен вибрати пункти меню «Перетворити в MBR» або

«Перетворити в GPT». При виборі цих пунктів програма запитує підтвердження на виконання дії, оскільки зміна типу завантажувального запису призводить до знищення всіх даних на диску.

Вимоги до продуктивності

Додаток може використовуватися на будь-якому обладнанні, не залежно від його потужності. Проте, продуктивність обладнання може впливати на швидкість виконання різних довготривалих операцій. При цьому програма має коректно реагувати на подібні операції.

Вимоги до безпеки

Додаток може коректно виконувати свою роботу тільки при наявності у користувача прав адміністратора, оскільки отримання інформації про диски і розділи, а також виконання різних дій над файловою системою вимагають наявність від докладання певних привілеїв.

2 Аналіз аналогів для розробки додатку

2.1 Файлові системи

Структура файлової системи FAT12 полягала в тому, що суміжні сектори диску об'єднувалися в спеціальні одиниці, які називаються кластерами. Кількість секторів має дорівнювати ступеню двійки. Для зберігання одного файлу повинно використовуватися ціле число кластерів. З цієї причини для того, щоб реальною інформацією займалася максимальна кількість секторів диску, необхідно мати мінімально можливий розмір кластера, а для скорочення обсягу адресної інформації і підвищення швидкості роботи з файлами розмір кластера повинен бути максимально великим. Для вирішення такого завдання в файловій системі використовується певний компроміс. Оскільки об’єм диску практично завжди не виражається цілим числом кластерів, в кінці тому зазвичай присутній залишок, розміром менше кластера, який не може відводитися операційною системою для зберігання інформації. У першій версії файлової системи FAT використовувалася 12-бітна адресація кластеру в 4 кілобайти. Це дозволяло їй підтримувати розділи до 16 мегабайтів. Уже тоді файлова система забезпечувала такі функції як ім'я файлу в форматі 8.3, каталоги, атрибути файлів, час створення і зміни файлів.

Також спеціально виділялася область кореневого каталогу. Вона мала строго фіксоване положення і фіксований розмір в 32 байтних елементах, тобто в таблиці записувалася саме кількість елементів, кожен з яких описував один з елементів кореневого каталогу (файл або інший каталог). Якщо кластер належить файлу, то його осередок в таблиці містить адресу наступного кластера цього файлу. Якщо осередок відповідає останньому кластеру файлу, вона містить спеціальне значення. Таким чином вибудовується ланцюжок кластерів файлу. Невикористані кластери в таблиці записуються нулями. Неробочі кластери (наприклад, через неможливість читання відповідної області пристрою) записуються в таблиці спеціальним кодом [1].

Важливим елементом файлової системи FAT є сама таблиця FAT. Ця таблиця займає окрему логічну область. Вона визначає набір кластерів, де розміщуються файли і папки тому. Безпосередньо після закінчення останньої таблиці слід обрати область даних, що містить файли і папки. Каталог FAT є звичайним файлом, який позначений спеціальним атрибутом [2].

Після файлової системи FAT з додаванням 16-бітної адресації і максимального розміру кластера в 32 кілобайти з'явилася файлова система FAT16, що підтримувала розділи до 2 гігабайт.

Надалі була представлена поліпшена версія файлової системи FAT16, звана FAT32. Створення її було обгрунтовано тим, що в FAT16 присутнє обмеження на розмір тому. Також нова версія файлової системи FAT дозволяла використовувати старий код програм MS-DOS, зберігаючи формат. FAT32 використовує 32-розрядну адресацію кластерів. Максимально можливе число кластерів в FAT32 дорівнює 268435445, що дозволяє використовувати логічні диски об'ємом до 8 терабайт. Розмір кластера за замовчуванням може змінюватися від 512 байт до 32 кілобайт в залежності від конкретної операційної системи і від розміру тому. Максимальний розмір файлу для даної файлової системи становить 4 гігабайт. Основною перевагою FAT32 на цей момент можна вважати те, що вона стала своєрідним стандартом і в багатьох носіях використовується за замовчуванням. Це необхідно для того, щоб носії могли підтримувати старі пристрої, що не мають можливості працювати з іншими файловими системами. FAT32 підтримується такими операційними системами

як Windows, Mac OS, Linux і т.д. Однак обмеження на розмір файлу та розмір є дуже істотним недоліком, що стало приводом для розробки нових файлових систем [2].

Вважається, що на заміну FAT прийшла файлова система NTFS. З метою поліпшення продуктивності, надійності і ефективності використання дискового простору для зберігання інформації про файли тут використовуються спеціалізовані структури даних. Інформація про файли зберігається в головній файловій таблиці - Master File Table (MFT). NTFS підтримує розмежування доступу до даних для різних користувачів і груп користувачів, а також дозволяє призначати дискові квоти (обмеження на максимальний обсяг для різних користувачів). Для NTFS розмір кластера за замовчуванням становить від 512 байт до 64 кілобайт в залежності від розміру тому і версії операційної системи. Специфікації файлової системи NTFS закриті. Це створює певні труднощі при реалізації її підтримки в продуктах, які не належать фірмі Microsoft, наприклад, розробникам драйверів для вільних операційних систем доводиться займатися зворотною розробкою файлової системи NTFS. В цей час повноцінна підтримка NTFS присутня тільки в операційних системах сімейства Windows від фірми Microsoft. Однак існують різні засоби для роботи з даною файловою системою в інших операційних системах. На цей момент NTFS вважається найбільш поширеною файловою системою [3].

Для зручності роботи з жорсткими дисками простір на них заведено ділити на так звані розділи. Виділення розділів зазвичай практикується на внутрішніх завантажувальних дисках комп'ютера, оскільки основною його метою є відділення файлів операційної системи від файлів користувача і від файлів інших операційних систем, що знаходяться на тому ж фізичному носії. Крім цього, така система має таку важливу перевагу як використання різних файлових систем на різних розділах одного жорсткого диска.

2.2 Типи завантажувального запису

Одним з найбільш поширених засобів подання списку розділів є головний завантажений запис або MBR (Master Boot Record). MBR містить код і дані, необхідні для подальшого завантаження операційної системи. Ці дані містяться на перших фізичних секторах жорсткого диска та містять невеликий фрагмент

коду, що виконується, таблицю розділів і спеціальну сигнатуру. Основна функція головного завантажувального запису - перехід в той розділ жорсткого диска, з якого надалі буде виконуватися подальший код (найчастіше - операційна система). У процесі запуску комп'ютера BIOS завантажує код MBR в оперативну пам'ять і передає управління в завантажувальний код MBR. У таблиці розділів міститься вся необхідна системі інформація про кількість розділів і їх стані. Є ознака активного розділу - це той розділ, якому надалі передається управління при завантаженні. У першому секторі кожного основного (активного) розділу знаходиться завантажувальний сектор, що відповідає за завантаження операційної системи з цього розділу. У головного завантажувального запису під таблицю розділу виділено 64 байти, кожен запис займає 16 байт. Таким чином, всього на жорсткому диску може бути створено не більше 4 розділів, що в момент розробки MBR вважалося достатнім. Проте, пізніше був введений додатковий (розширений) розділ, який сам не мав файлової системи, але містив в собі інші логічні розділи. Така структура дозволяла позбутися обмежень в кількості розділів.

Крім MBR існує інший стандарт формату розміщення таблиць розділів, іменований GUID Partition Table (GPT). Він є частиною розширюваного мікропрограмного інтерфейсу (EFI) - стандарту, запропонованого Intel на зміну BIOS. EFI використовує GPT там, де BIOS використовує головний завантажувальний запис. На відміну від MBR, GPT спирається на розширені можливості EFI для здійснення процесів ініціалізації завантаження активного розділу [4]. Кількість розділів не обмежена і залежить тільки від операційної системи. Крім того, GPT забезпечує дублювання - зміст і таблиця розділів записані як на початку, так і в кінці диска. Теоретично, GPT дозволяє створювати розділи диску розміром до 9,5 зетабайт, в той час як MBR може працювати тільки до 2,2 терабайт. GPT дозволяє призначати розділам ідентифікатори GUID, імена та атрибути, незалежно від внутрішніх UUID файлових систем, їх міток і іншого, щоб дозволити Вам отримувати за такими назвами замість менш зручних міток і номерів розділів. Завдяки підтримці Юнікоду в назвах і сприятливих обмежень на них, розділи можуть бути названі будь-якою мовою і згруповані по папках. Зміст таблиці розділів вказує ті логічні блоки на диску, які можуть бути залучені

користувачем. Воно так само вказує число і розмір записів даних про розділи, що становлять таблицю розділів. Стандартно в Microsoft Windows резервується 128 записів даних про розділи.

Таким чином можливе створення 128 розділів на диску. З недоліків такого підходу можна відзначити відсутність можливість призначати назву всьому диску, як окремих розділів, сильна прив'язка розділу до його номеру в таблиці, кількість дублікатів таблиць строго обмежена двома екземплярами, що в разі пошкодження даних недостатньо для їх відновлення.

На цей момент жодна файлова система і жоден формат завантажувальних записів не використовується повсюдно. У зв'язку з цим виникають неминучі проблеми з сумісністю. Незважаючи на те, що сучасні операційні системи підтримують більшість популярних файлових систем, можуть виникнути складнощі при обміні даними між пристроями з різними файловими системами. Через виникнення подібних проблем пересічному користувачеві комп'ютера часом не вдається абстрагуватися від нутрощів файлової системи, з якої працює його операційна система. Тому нерідкі ситуації, коли користувачеві необхідно мати можливість керувати розміткою жорсткого диска, управляти файловими системами на його розділах і іноді змінювати тип завантажувального запису. У такому випадку використовуються спеціальні програмні утиліти, спрямовані на взаємодію з жорстким диском в рамках набору інструментів, представлених операційною системою.

2.3 Можливі помилки та способи їх усунення

У більшості місць, в яких може статися помилка, використовується механізм «try-catch». При виникненні помилки в блоці «try», станеться перехід до блоку «catch» для її обробки. В ході обробки виводиться діалогове вікно, в якому користувач може вибрати окрему дію, яку необхідно провести для усунення неполадки [5].

При виклику функцій з бібліотеки операційної системи можуть виникати помилки виконання. Ці помилки не будуть викликати виключення в керованому коді. Обробка таких помилок здійснюється так само, як і в мові C - через отримання коду помилки і його розшифровку.

Всі помилки можна поділити на дві групи: помилки, пов'язані з роботою

програми та помилки, не пов'язані з роботою програми. Помилки функцій бібліотеки операційної системи заведено вважати помилками, не пов'язаними з роботою додатка. По кожній групі помилок буде описано детально в наступних пунктах.

Помилки, які не пов'язані з роботою програми, - це помилки, виникнення яких не пов'язане з роботою програми, і вони виникають з об'єктивних причин. Деякі з них можливо обробити, деякі - ні. Для виправлення помилок не потрібно змінювати вихідний код програми з подальшою перекомпіляцією.

В основному подібні помилки можуть виникнути при виконанні функцій бібліотеки. Неможливо заздалегідь передбачити дії бібліотеки при виклику її функцій додатком, тому після виклику кожної функції відбувається перевірка на коректність її виконання. У разі некоректного виконання за допомогою спеціальних методів виходить код помилки. Користувачеві показується вікно з розшифровкою такого коду помилки.

Додатковим місцем виникнення помилки може бути робота графічних компонентів програми. При використанні різних версій графічних бібліотек, поведінка компонентів може незначно відрізнятися. При цьому можливе виникнення помилок, які будуть оброблені в додатку блоками «try» і «catch», при спрацьовуванні яких будуть виконуватися необхідні дії [6].

Можливе виникнення помилки завантаження модуля. Програми, написані на мові C#, підвантажують модулі після першого звернення до них. Всі модулі зберігаються на диску у вигляді байт коду, який частково скомпільовано і вимагає остаточної компіляції при використанні. У разі, якщо компілятор не знаходить бібліотеку, виконання програми зупиняється. У цьому випадку необхідно перевірити наявність бібліотеки на диску (її назву буде виведено в повідомленні про помилку) і перезапустити програму.

Помилки, пов'язані з роботою програми, - це помилки, виникнення яких відбувається через помилки в вихідному коді програми. Для виправлення помилок не потрібно змінювати вихідний код програми і перекомпілювати додаток. Але також існують можливості обходу помилки і виконання необхідного дії іншим шляхом.

Всі програмісти при роботі припускаються помилок, кожен день знаходять

помилки в багатьох Програмних продуктах. Говорити про наявність у додатку помилок не можна, але є достатня ймовірність їх прояву.

NullReferenceException - найпопулярніший тип виключення, що виникає в мові C#. Зазвичай велика частина помилок в прикладних програмах мають саме цей тип або випливають з неправильної обробки даної ситуації. Дана помилка означає, що програма звернулася до неіснуючого об'єкту. Через часте виникнення подібних помилок, в операційні системи була додана від таких помилок - перші сторінки адрес в пам'яті зарезервовані і при зверненні до них генерується виключення. Дані помилки відбуваються в більшості випадків з вини програміста через неуважність. Їх виправлення відбувається швидко, тому для якнайшвидшого виправлення необхідно звернутися до програміста із зазначенням дій по відтворенню та, по можливості, дампом додатку.

При зверненні до елементу призначеного для користувача інтерфейсу з іншої нитки за допомогою механізму Invoke, може статися помилка TargetInvocationException, яка означає, що під час роботи механізму Invoke, відбулося виключення. Найчастіше дана помилка відбувається при зверненні до елементу призначеного для користувача інтерфейсу після того, як він був знищений. В цьому випадку виключення ігнорується, оскільки дані вже не потрібні, якщо користувач закрив додаток або видалив елемент, в який вони повинні були бути додані.

2.4 Мови програмування високого рівня

Наразі у середовищі розробників вважається, що мови програмування, які мають прямий доступ до пам'яті та регістрів або мають асемблерні вставки, потрібно вважати мовами програмування з низьким рівнем абстракції.

Розглянемо декілька з них:

Java — об'єктно-орієнтована мова програмування, випущена 1995 року компанією «Sun Microsystems» як основний компонент платформи Java. З 2009 року мовою займається компанія «Oracle», яка того року придбала «Sun Microsystems». В офіційній реалізації Java-програми компілюються у байт-код, який при виконанні інтерпретується віртуальною машиною для конкретної платформи.

«Oracle» надає компілятор Java та віртуальну машину Java, які

задовольняють специфікації Java Community Process, під ліцензією GNU General Public License.

Мова значно запозичила синтаксис із C і C++. Зокрема, взято за основу об'єктну модель C++, проте її модифіковано. Усунуто можливість появи деяких конфліктних ситуацій, що могли виникнути через помилки програміста та полегшено сам процес розробки об'єктно-орієнтованих програм. Ряд дій, які в C/C++ повинні здійснювати програмісти, доручено віртуальній машині. Передусім Java розроблялась як платформонезалежна мова, тому вона має менше низькорівневих можливостей для роботи з апаратним забезпеченням, що в порівнянні, наприклад, з C++ зменшує швидкість роботи програм. За необхідності таких дій Java дозволяє викликати підпрограми, написані іншими мовами програмування.[11]

Turbo, а пізніше Borland Pascal— це одна з найвдаліших та найпоширеніших реалізацій мови Pascal, створена компанією Borland. Turbo Pascal — розширення американського стандарту (ANSI Pascal), яке враховує архітектурні особливості MS-DOS та MS Windows і постачається зі значними за обсягом і різноманітності пакетами стандартних процедур. Такі принципові нововведення, як апарат модулів і об'єктно-орієнтовані засоби полегшують конструювання великих програмних систем на основі технології модульного програмування.

Компілятор Turbo Pascal працює за однопрохідною схемою, реалізує функції редагування зв'язків, формуючи на виході готовий до виконання об'єктний код. Компілятор може здійснювати широкий набір локальних оптимізацій (згортання констант, виключення невикористовуваного коду і зайвих даних, оптимізація операцій і т. д.), що сприяє високій ефективності кінцевих програм.

Система Turbo Pascal є інтегрованим середовищем (IDE), яке налічує ряд компонентів, що в сукупності підтримують усі види робіт зі створення програм. Система містить універсальний текстовий редактор, компілятор вхідної мови, редактор зв'язків і вбудований символьний зневаджувач. Багатовіконний інтерфейс із розвинутою системою меню і досконалою довідковою системою забезпечує високу продуктивність праці програміста.[12]

C# — об'єктно-орієнтована мова програмування з безпечною системою типізації для платформи .NET. Розроблена Андерсом Гейлсбергом, Скотом Вілтамутом та Пітером Гольде під егідою Microsoft Research (належить Microsoft).

Синтаксис C# близький до C++ і Java. Мова має строгу статичну типізацію, підтримує поліморфізм, перевантаження операторів, вказівники на функції-члени класів, атрибути, події, властивості, винятки, коментарі у форматі XML. Перейнявши багато від своїх попередників — мов C++, Object Pascal, Модула і Smalltalk — C#, спираючись на практику їхнього використання, виключає деякі моделі, що зарекомендували себе як проблематичні при розробці програмних систем, наприклад, мова C#, на відміну від C++, не передбачає множинне успадкування класів.[13] Саме це і стане основною причиною вибору C#.

2.5 Аналоги додатків

EaseUS Partition Master одна з найпопулярніших програм для управління розділами жорстких дисків в Windows. Забезпечує виділення нових розділів без втрати даних.

Переваги EaseUS Partition Master:

- Просте перетворення типу розділу, основного на логічний і навпаки
- Відновлює видалені або неіснуючі розділи
- Підтримувана місткість жорсткого диска до 8 ТБ

Недоліки:

- Відсутність можливості перенесення даних з HDD на SSD в безкоштовній версії.

AOMEI Partition Assistant також користується популярністю. Створює, розділяє, з'єднує, копіює розділи жорстких дисків, змінюючи їх розмір, зберігаючи при цьому файли. Можливе перенесення системи.

Переваги:

- Зручні майстри для кожної операції
- Підтримує всі найпопулярніші файлові системи
- Відображає точну інформацію про підтримуваний носій
- Дозволяє створити завантажувальний диск CD з додатком.

Gparted інструмент для управління розділами жорсткого диска комп'ютера. Поширюється у вигляді ISO-файла. Встановіть його на флешку або запишіть на

компакт-диск і запустіть з нього комп'ютер.

Переваги Gparted:

- Не вимагає установки і не займає місця на диску
- Привабливе безкоштовна пропозиція для компаній
- Широкі можливості управління розділами практично будь-якої файлової системи
- Зручний візуальний майстер для кожної операції
- Дозволяє конвертувати NTFS в FAT32 без форматування, змінювати тип диска з MBR на GPT

Недоліки:

- Освоєння вимагає часу
- Доступний тільки за допомогою Live CD MiniTool Partition Wizard
- Багато інструментів, в тому числі для очищення і перевірки диска

Partition Manager виконує операції з розділами диска. Форматує флешку в FAT32 і NTFS. Виправляє MBR-диски. Перетворює MBR в GPT і навпаки.

Переваги:

- Інструмент для створення образів дисків
- Зручний майстер створення і зміни розділів
- Вбудований редактор boot сектора, дозволяє вручну вносити зміни
- Показує дані SMART для жорстких дисків

Вже існуючі рішення потребують значних коштів для користування та їх підтримки з третьої сторони, тому було обрано рішення розробити його самостійно у рамках проєкту. Найбільш зручними були обрані:

a) Файлова система FAT32
b) Мова програмування C#
c) Середовище розробки Microsoft Visual Studio яке підтримує мову C#

3 Вибір технології та методу розробки

3.1 Мова програмування C#

C# - об'єктно-орієнтована мова програмування. Дана мова має великі функціональні можливості, наприклад підтримка поліморфізму, перевантаження операторів (в тому числі операторів явного і неявного приведення типу) делегати, властивості, ітератори, атрибути, узагальнені типи і методу, анонімні функції, виняток. З обмежень

мови можна виділити відсутність підтримки множинного спадкоємства (допускається множинна реалізація інтерфейсів).

Головною причиною вибору даної мови є платформа .NET, що володіє великою кількістю бібліотек, що дозволяють значно полегшити розробку кінцевого продукту, а також наявність готових бібліотек для розробки графічного інтерфейсу [7].

Використання мови C # дозволяє за допомогою спеціальних бібліотечних компонентів викликати зі свого додатка функції з API операційної системи, написані на мові C. Таким чином можлива розробка керованої об'єктно-орієнтованої бібліотеки для роботи з файловою системою, яка всередині себе буде працювати з функціями операційної системи.

3.2 Microsoft Visual Studio

Microsoft Visual Studio - інтегроване середовище розробки (IDE), розроблене компанією Microsoft, що містить в собі велику кількість інструментів для розробки програмних продуктів різної складності. Дане середовище розробки підтримує велику кількість мов програмування, таких як C#, C, C ++ і багато інших. Зручний механізм налагодження додатків, багатофункціональний текстовий редактор з технологією автодоповнення IntelliSense робить Visual Studio ідеально відповідним для вирішення поставленого завдання і одним з найбільш зручних засобів розробки на сьогодні [8].

Проєктування додатка здійснювалося методом спадного програмування. При цьому методі проєктування здійснюється зверху вниз. Спочатку пишеться головний модуль, а спадкові функції обробки даних замінюються заглушками. Далі налагоджується головний модуль і

заглушки замінюються конкретними модулями для обробки даних. Цей процес повторюється до тих пір, поки не будуть налагоджені модулі найнижчих рівнів. Перевагою такого підходу при написанні програм є те, що процес написання програми збігається з процесом налагодження.

При розробці було вирішено використовувати метод TDD (Test Drive Development), згідно з яким спочатку пишеться тест, що покриває бажані зміни, потім пишеться код, що дозволяє пройти тест, потім проводиться рефакторинг написаного коду та цикл повторюється.

Роботу зі створення додатка, умовно можна розбити на кілька етапів:

– попередній етап;

– розробка архітектури проєкту;

– програмування;

– розробка дизайну додатку;

– тестування;

– розміщення;

– адміністрування (підтримка).

Кожен етап охоплює велику частину роботи, після якого є певний доробок, повертатися до якого вже не треба. Даний принцип розробки можна віднести до методології WaterFall або каскадної моделі.

Каскадна модель (англ. Waterfall model, іноді перекладають, як модель «Водоспад») - модель процесу розробки програмного забезпечення, в якій процес розробки виглядає як потік, що послідовно проходить фази аналізу вимог, проєктування, реалізації, тестування, інтеграції та підтримки.

Дотримуючись каскадної моделі, розробник переходить від однієї стадії до іншої строго послідовно. Спочатку повністю завершується етап «Визначення вимог», в результаті чого виходить список вимог до ПО. Після того як вимоги повністю визначені, відбувається перехід до проєктування, в ході якого створюються документи, що детально описують для програмістів спосіб і план реалізації зазначених вимог. Після того як проєктування повністю виконане, програмістами виконується реалізація отриманого проєкту. На наступній стадії процесу відбувається інтеграція окремих компонентів, що розробляються різними командами програмістів. Після того як реалізація і

інтеграція завершені, проводиться тестування і налагодження продукту; на цій стадії усуваються всі недоліки, що з'явилися на попередніх стадіях розробки. Після цього програмний продукт впроваджується і забезпечується його підтримка - внесення нової функціональності і усунення помилок.

Тим самим, каскадна модель має на увазі, що перехід від однієї фази розробки до іншої відбувається тільки після повного і успішного завершення попередньої фази, і що переходів назад або вперед або перекриття фаз не відбувається.

Проте, існують модифіковані каскадні моделі (включаючи модель самого Ройса), що мають невеликі або навіть значні варіації описаного процесу.

4 Проєктування програми

4.1 Попередній етап

На попередньому етапі встановлюються всі необхідні програмні засоби і створюється каркас додатку. Також підключаються всі необхідні бібліотеки.

Спочатку необхідно встановити Visual Studio. На офіційному сайті є дистрибутив Community Edition. Для цілей даної дипломної роботи його цілком вистачає.

Каркас додатка створювався за допомогою майстра створення нових додатків Visual Studio. Був обраний тип проєкту Windows Forms для мови C#. Структура програми та залежності стандартні для даного типу проєктів.

4.2 Розробка архітектури проєкту

Для організації архітектури проєкту був обраний патерн проєктування MVC. Згідно з яким, необхідно розділити компоненти, що відповідають за візуальне уявлення, логіку програми та моделі даних. Більшість сучасних додатків в тій чи іншій мірі реалізують даний патерн проєктування. Цьому існує декілька причин, головною з яких є можливість реалізації більш складних архітектурних рішень меншими засобами. Також при використанні даного патерну проєктування існує чітке розмежування обов'язків між об'єктами, внаслідок чого локалізація помилок стає простіше.

Додатковим архітектурним рішенням було використання ООП в розробці програми. Весь додаток повинен складатися з класів (об'єктів), які матимуть

властивості (стану) і надавати користувачеві певний інтерфейс (поведінку). В основі ООП лежать три основних принципи: інкапсуляція, поліморфізм, успадкування. У розробці програми будуть виконані всі перераховані принципи. Як говорилося вище, класи будуть надавати властивості і інтерфейс, а принцип інкапсуляції допоможе приховати внутрішні змінні і методи класу від користувача, оскільки вони йому не потрібні, а неконтрольований виклик даних методів може привести до небажаних наслідків. Спадкування пронизує додаток повністю. Всі форми розробки будуть успадковані від класу Form, а класи, необхідні для роботи з файловою системою, від інтерфейсу IDisposable. Дані класи будуть являє собою об'єкти, що моделюють реальні диски і розділи. Спадкування даних класів від інтерфейсу IDisposable необхідно для закриття описувача диску або розділу при утилізації об'єкта. Поліморфізм в додатку, що розробляється має менше значення, але все одно використовується. При показі діалогового вікна або будь-якої форми в якості родового їй передається об'єкт класу Control, хоча насправді в додатку цього параметра будуть передаватися об'єкти класу, успадкованого від класу Form, який через кілька батьків має саме цей клас.

Компонентами вистави в додатку, що розробляється будуть файли з розширенням «.designer.cs». Дані файли автоматично генеруються дизайнером форм. В них розходиться інформація про розмітку форми, всі її компоненти, їх властивості та події, які необхідно обробити.

Контролером в додатку, що розробляється будуть файли, що мають ту ж назву, що і файли уявлення, але мають розширення «.cs». В даному файлі знаходиться обробка подій інтерфейсу користувача, а також готуються об'єкти моделей для відображення на екрані.

Об'єктами моделей в даному додатку будуть всі інші файли коду з розширенням «.cs». В них буде реалізована об'єктна модель додатку. Основними елементами графічної частини програми при цьому будуть об'єкти, успадковані від класу Form, які являють собою різні дочірні вікна основного додатка, які надають користувачу можливість вибору різних параметрів виконуваних над файловою системою операцій. Схема взаємодії системи представлена на Рис 2.

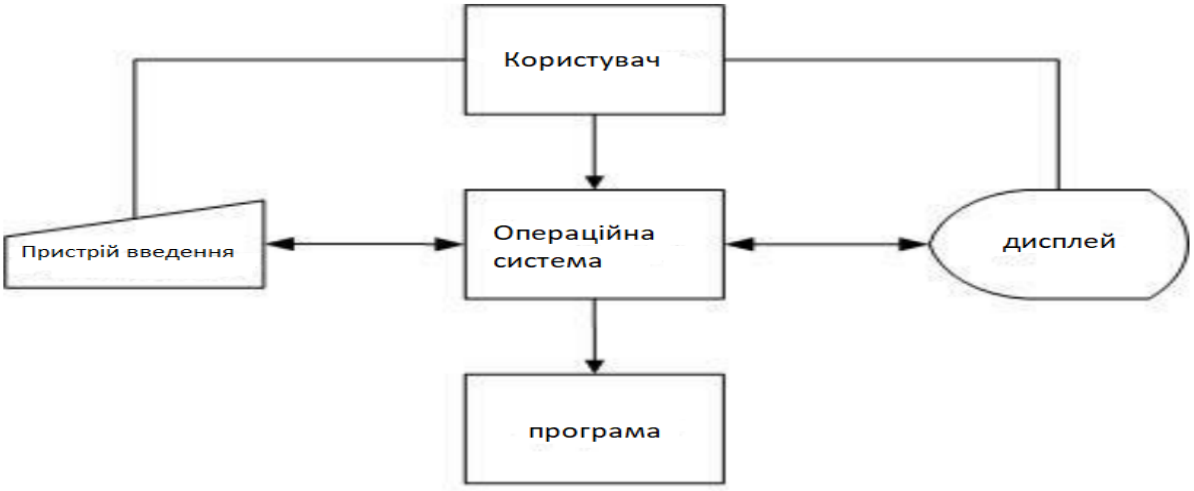

Рисунок 2 - Схема взаємодії системи

Операційна система обробляє виклики програми. Користувач взаємодіє з операційною системою, яка обробляє його запити від пристроїв введення та передає програмі. Операційна система є центральною точкою в даній схемі. Вона обробляє всі запити, зокрема виводить на дисплей зображення, підготовлене додатком для демонстрації користувачеві.

4.3 Проєктування бібліотеки взаємодії з розміткою

Бібліотека взаємодії з розміткою жорсткого диска повинна задовільняти вимоги до подібного роду бібліотек, що передбачає повторне використання цього додатка. Методи і класи даної бібліотеки повинні бути зрозумілими і логічними, вони повинні супроводжуватися коментарями, які описують їх функціонал. Оскільки дві основних сутності при роботі з файловою системою - це диски і розділи, то двома основними класами для управління розміткою будуть Disk і Partition. Архітектура операційної системи передбачає наявність у кожного диску і розділу свого описувача, який використовується для подальшого виклику функцій з API операційної системи. Передбачуваний варіант реалізації бібліотеки представлений на Рис 3.

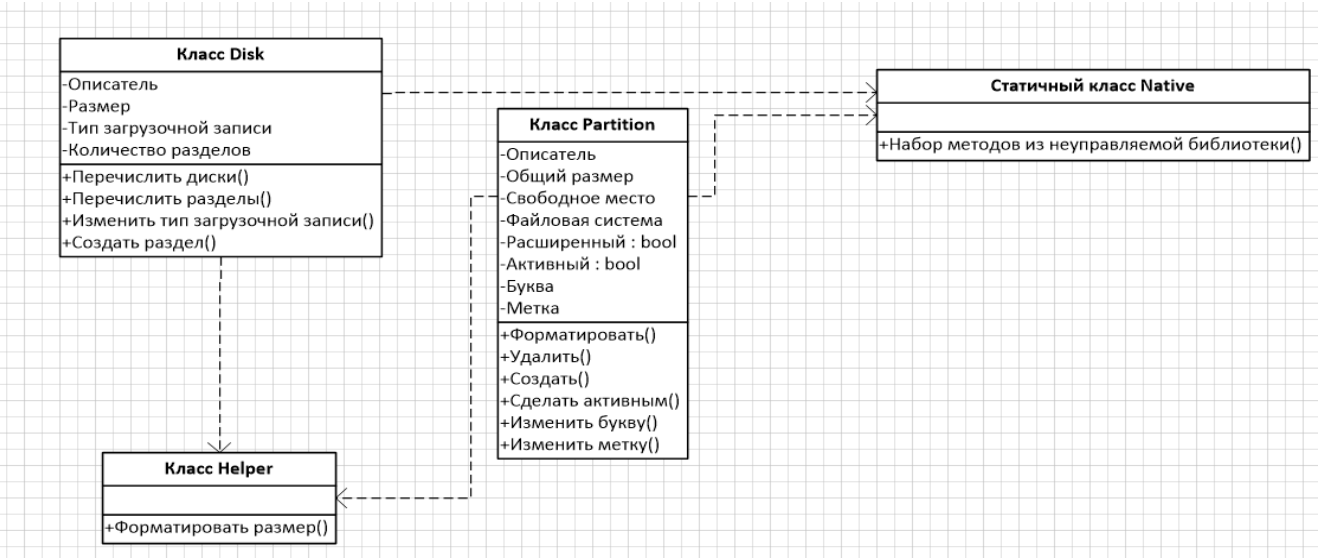

Рисунок 3 - Модель бібліотеки взаємодії з розміткою жорсткого диску

Окрім двох основних класів по роботі з розміткою, необхідний також статичний клас для маршаллінгу функцій некерованого коду в керований код. Методи класів Disk і Partition викликатимуть всередині себе статичні методи цього класу. Також для різних перетворень розміру і деяких функцій, що виконують простий функціонал і концептуально не входять ні в один із запропонованих класів буде використовуватися окремий статичний клас Helper.

4.4 Проєктування графіку дисків і розділів

Для проєктування графіку дисків і розділів створимо клас GraphControl і успадкуємо його від класу Control графічної бібліотеки. Для рисування графіку скористаємося методами простору імен System.Painting графічної бібліотеки. При створенні даний компонент в конструкторі повинен приймати список дисків і розділів, аналізуючи які і буде будуватися графік. В процесі роботи повинна бути можливість змінити дані списки ззовні, тим самим перерисувавши графік на актуальний. Даний елемент так само повинен володіти функціоналом для взаємодії із зовнішніми класами за допомогою подій, які будуть викликатися при кліці на різні елементи графіка. При рисуванні графіку для кожного диску береться весь список розділів, загальний розмір диску, обчислюється частина простору, яка займається розділом, і рисується область даного розділу кольором, відповідному порядковому номеру розділу. Алгоритм рисування графіка

представлений на Рис 4.

Рисунок 4 - Алгоритм рисування графіка

Як бачимо з представленого алгоритму, рисування графіка буде здійснюватися за допомогою двох вкладених циклів. Зовнішній цикл буде являти собою перерахування всіх дисків. На кожній ітерації цього циклу будуть ставитися необхідні відступи і рисуватися новий рядок з назвою відповідного диску. Потім викликається вкладений цикл з перерахуванням всіх розділів, що знаходяться на поточному жорсткому диску. Для кожного з них буде рисуватися відповідна частина графіка.

5 Реалізація програми

5.1 Опис основних класів додатку

Точкою входу додатку, як і будь-якої програми, написаної на мові C#, є статична функція Main, розташована в статичному класі Program. У даній функції створюється основна форма додатку, проводиться її відображення методом Show (). Потім в функції Main викликається статичний метод віконної бібліотеки Application.Run (). Даний метод повертає управління тоді, коли всі графічні вікна програми відкриті. Після повернення даного методу буде завершена і сама програма. Функціонал роботи основної форми додатку реалізований в класі MainForm. З цього класу в діалоговому режимі відкриваються всі інші форми додатків. Повний список форм додатки показаний в таблиці 6.

Таблиця 6 - Форми додатку

Ім'я класу	Короткий опис
MainForm	Основна форма додатку. Відображає основну інформацію про диски і розділи, дозволяє взаємодіяти з функціоналом додатку.
CreateForm	Форма створення розділу. Дозволяє користувачеві вказати параметри створюваного розділу і викликає функцію створення.
ChangeLabelForm	Форма зміни мітки розділу. Дозволяє змінити мітку переданого параметром розділу.
ChangeLetterForm	Форма зміни букви розділу. Дозволяє змінити букву переданого параметром розділу.
CheckPartitionForm	Форма перевірки розділу. Викликає в окремій нитці функцію перевірки переданого розділу.
FormatForm	Форма форматування розділу. В окремій нитці виробляє форматування розділу із заданими користувачем параметрами.
PartitionInformationForm	Форма властивостей розділу. Відображає властивості переданого в якості параметру розділу.

Як бачимо з наведеної таблиці, програма має велику кількість вбудованих вікон, оскільки можливі операції, які користувач буде здійснювати з додатком, вимагають введення від користувача великої кількості параметрів. Так, наприклад, форма форматування вимагає вибрати від користувача файлову систему, в яку буде здійснено форматування, а також, чи потрібно користувачеві швидке форматування. Форма створення розділу вимагає від користувача введення розміру створюваного розділу.

Крім подібних форм, є також форми для відображення властивостей диску або розділу. Дана форма не вимагає введення даних від користувача і являє собою звичайне інформаційне вікно. Код форми є простим заповненням полів тієї інформації, яка виходить за допомогою спеціальної функції з бібліотеки. Повна діаграма класів представлена на Рис 5.

Рисунок 5 - Діаграма класів

Крім основних класів-форм додатку, в програмі використовуються також і додаткові допоміжні класи. Клас Resource використовується для представлення ресурсного файлу програми і дозволяє використовувати зображення що зберігаються в цьому файлі. Клас Sorting реалізує сортування дерева дисків і розділів на основній формі додатку.

5.2 Реалізація основної форми додатку

Основна форма додатку служить для взаємодії користувача з усім можливим функціоналом додатку, а також відображає основну інформацію про диски і розділи. У верхній частині форми знаходиться рядок головного меню програми. Нижче розташований список дисків і розділів, що має деревоподібну структуру. Під деревоподібним списком розташовується графік дисків і розділів, який реалізує графічне відображення файлової системи. На таблиці 7 представлені методи головної форми додатку.

Таблиця 7 - Методи головної форми додатку

Ім'я	Повертається тип	Опис
HideTools	Void	Видалення інструментів з панелі
ShowTools	Void	Відображення елементів панелі інструментів в залежності від обраного в дереві елемента
InitGraph	Void	Створення графіка за новими даними про диски і томи
FillDisksInfo	Void	Заповнення дерева дисків і томів
AddVolumeNodes	Void	Додавання елементів в дерево для кожного диску і розділу
GetImageOfColor	Image	Отримати зображення з переданим кольором
ConvertPartitionStyleToString	String	Перетворює тип розмітки в назву цього типу розмітки
DeletePartition	Void	Видаляє вибраний розділ

Продовження таблиці 7

Ім'я	Повертається тип	Опис
ChangeLetter	Void	Змінити букву обраного розділу
DeleteLetter	Void	Видалити букву розділу
ChangeLabel	Void	Змінити мітку розділу
FormatPartition	Void	Форматувати розділ
CreateParitition	Void	Створити розділ
CheckPartition	Void	Перевірити розділ
WriteLoadBootCode	Void	Оновити завантажувальний код розділу
ShowPartitionProperties	Void	Відобразити властивості розділу
WriteLoadCode	Void	Записати завантажувальний код
ConvertDiskToGpt	Void	Перетворити диск в GPT
ConvertDiskToMbr	Void	Перетворити диск в MBR

Кінцевий вигляд основної форми програми показаний на Рис 6.

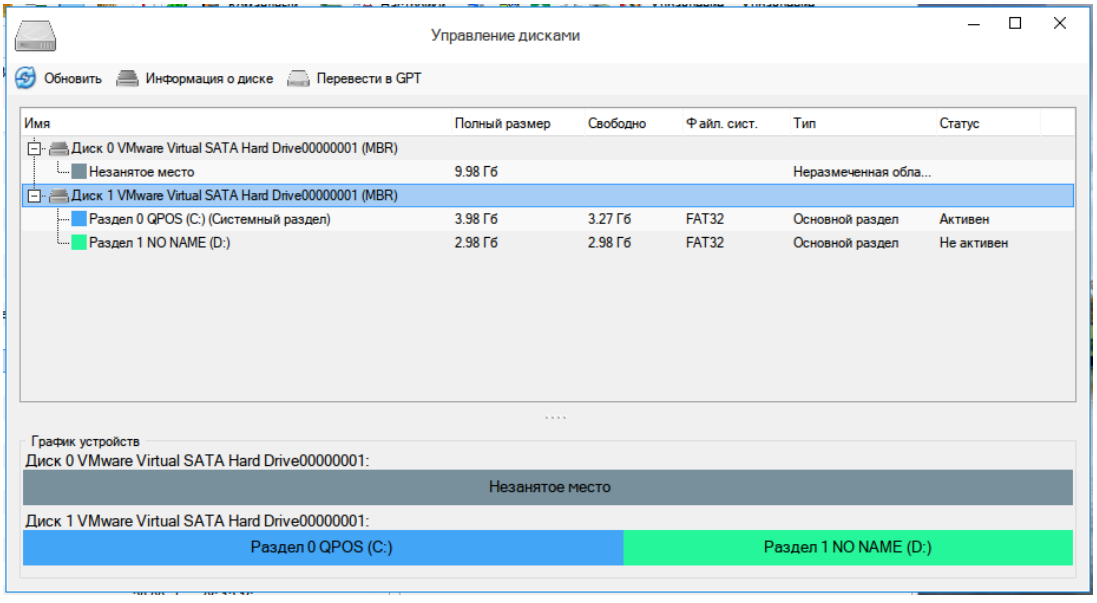

Рисунок 6 - Основна форма програми

Заповнення списку здійснюється за допомогою перерахування дисків і розділів спеціальними функціями бібліотеки роботи з файловою системою. Код заповнення списку представлений нижче.

```
/// <summary>
/// Заполнение дерева
/// </summary>
private void AddVolumeNodes()
{
    _disksList = Disk.Enumerate(false);
    foreach (var newDisk in _disksList)
    {
        Color colorOfParttion = Color.FromArgb(0x42,
        0xa5, 0xf5);
        var tli = new TreeListItem(diskList)
        {
            Image = Resource.prop,
            Tag = newDisk,
            Text = newDisk.m_DiskName +
            String.Format(" ({0})",
        ConvertPartitionStyleToString(newDisk.PartStyle)),
        }; //Итем диска
        tli.AfterCollapse += tli_AfterCollapse;
        if (newDisk.DiskInformation.m_MediaType ==
        Cryptosoft.Qpos.DiskApi.MEDIA_TYPE.RemovableMedia)
        {
            tli.Image = Resource.usb_16px;
        }
        diskList.Items.Add(tli);
        tli.Focus();
        var partitions = newDisk.m_Partitions;
        foreach (var partition in partitions)
        {
            bool isFree = false;
            bool isSystem =
            _sSystemPath.StartsWith(partition.Letter.ToString());
            var tlci = new TreeListItem(diskList);
            tlci.ImageSize = new Size(13, 13);
            if (partition.PartName == FREE_SPACE)
            {
                tlci.Text = FREE_SPACE;
                isFree = true;
            }
            else
            {
                tlci.Text = partition.PartName;
                if (isSystem)
                    tlci.Text += " (Системный раздел)";
            }
            if (partition.PartType != FILE_SYSTEM_FREE
            && partition.PartType != EXT_PARTITION)
            {
                tlci.Image =
                GetImageOfColor(colorOfParttion);
            }
            else
            {
                tlci.Image = Resource.color5;
            }
            colorOfParttion =
            DiskGraphControl.GetNextColor(colorOfParttion);
            //заполнение сабитемов для каждого тома
            tlci.Tag = partition;
            var tlcsi = new TreeListSubItem(tlci, 101)
```

```
            {
                Text = partition.FullSize,
            };
            var tlcsi4 = new TreeListSubItem(tlci, 104)
            { Text = partition.PartType };
            if (!isFree)
            {
                var tlcsiFreeSpace = new
                TreeListSubItem(tlci, 102)
                { Text = partition.FreeSpaceSize };
                var tlcsiFileSystem = new
                TreeListSubItem(tlci, 103)
                { Text = partition.FileSystemName };
                var tlcsiActivity = new
                TreeListSubItem(tlci, 105);
                if (partition.Active)
                {
                    tlcsiActivity.Text = "Активен";
                }
                else
                {
                    tlcsiActivity.Text = "Не активен";
                }
            }
            //_volumesList.Add(di);
            _partitionsList.Add(partition);
            tli.Items.Add(tlci);
        }
        tli.Expanded = true;
    }
}
```

Як бачимо з коду заповнення, отримання списку дисків здійснюється за допомогою бібліотечного класу Disk і його методу Enumerate (). Даний метод повертає список всіх знайдених операційною системою дисків. Потім здійснюється прохід в циклі по отриманому списку. Для кожного диску створюється елемент дерева (об'єкт типу TreeListItem), заповнюються колонки списку. Потім виходить масив розділів для диску через властивість m_Partitions. Для розділів виконуються схожі операції, але для елемента дерева, відповідного розділу, батьком вказується елемент диска. Для розділу заповнюються колонки з інформацією про розмір, вільне місце, тип файлової системи і т.д. Кожному елементу розділу призначається колір, аналогічний кольору в графіку, для зручної навігації.

Динамічне відображення елементів головного меню реалізовано за допомогою приховування неактуальних в цей час елементів меню за допомогою властивості Visible. Метод перевірки актуальних пунктів меню викликається кожен раз при виділенні елемента в списку дисків і розділів. Код зміни меню представлений нижче.

```
/// <summary>
/// Отображение элементов панели инструментов в зависимости от выбранного в дереве элемента
```

```
    /// </summary>
    private void ShowTools()
     {
          tiRefr.Visible = true;
    //отображение элементов панели инструментов аналогично отображению пунктов меню при вызове
контекстного меню
    var target = diskList.SelectedItem as TreeListItem;
          if (target == null)
              return;
          if (target.Tag is Disk)
          {
              var tag = target.Tag as Disk;
          tiProp.Visible = true;
              if (tag.PartStyle ==
              PARTITION_STYLE.PARTITION_STYLE_GPT)
              {
                  tiToMbr.Visible = true;
              }
              else if (tag.PartStyle ==
              PARTITION_STYLE.PARTITION_STYLE_MBR)
              {
                  tiToGpt.Visible = true;
              }
              else if (tag.PartStyle ==
              PARTITION_STYLE.PARTITION_STYLE_RAW)
                  // диск с разметкой raw можно преобразовать в любую другую разметку
              {
              tiToGpt.Visible = true;
              tiToMbr.Visible = true;
              }
              //tiWriteLoadCode.Visible = true;
          }
          else if (target.Tag is Partition && target.Text !=
          FREE_SPACE)
          {
              var vi = (Partition)target.Tag;
              if (!vi.Active)
              {
                  tiMakeActive.Visible = true;
              }
              else
              {
                  tiUndoActive.Visible = true;
              }
              tiChangeLabel.Visible = true;
              if
              (!_sSystemPath.StartsWith(vi.Letter.ToString()) || vi.Letter == '\0')
              {
                  tiChangeLetter.Visible = true;
                  tiDelete.Visible = true;
                  tiFormat.Visible = true;
              if (vi.FileSystemName ==
              FILE_SYSTEM_FAT32)
                  {
                  }
              }
          }
          else if (target.Text == FREE_SPACE)
          {
              var vi = (Partition)target.Tag;
              if (vi.PartType == EXT_PARTITION)
              {
                  tiCreate.Visible = true;
                  bool isLogic = false;
                  foreach (var volume in _partitionsList)
                  {
                      if (volume.sDisk.Handle ==
                      vi.sDisk.Handle)
                      {
```

```
                    if (volume.PartType ==
                    LOGICAL_PARTITION)
                    {
                        isLogic = true;
                    }
                }
            }
            if (vi.PartType == EXT_PARTITION)
            {
                if (!isLogic)
                    tiDelete.Visible = true;
            }
        }
        else
        {
            int partitionCount = 0;
            foreach (var volume in _partitionsList)
            {
                if (volume.sDisk.Handle ==
                vi.sDisk.Handle)
                {
                    if (volume.PartType !=
                    LOGICAL_PARTITION)
                    {
                        if (volume.PartType !=
                        FILE_SYSTEM_FREE)
                            partitionCount += 1;
                    }
                }
            }
        if (vi.sDisk.PartStyle ==
        PARTITION_STYLE.PARTITION_STYLE_MBR)
            {
                if (partitionCount < 4)
                    tiCreate.Visible = true;
            }
            else
            {
                tiCreate.Visible = true;
            }
        }
    }
}
```

Для кожного пункту меню перевіряється якась умова його відповідності виділеному в цей час елементу. Деякі пункти меню актуальні тільки якщо обраний диск. У їх числі зміни типу завантажувальної таблиці. Для розділів набір можливих функцій ширше. Для активного розділу відображається пункт меню «Прибрати активність», для неактивного - «Зробити активним». Пункт меню «Змінити мітку розділу» доступний для будь-якого розділу, крім розділу з нерозміченою областю. Зміна літери недоступна для системного розділу і нерозміченою областю. Для цих же розділів є форматування. Для нерозміченої області і для розширеного диску доступний пункт меню «Створити розділ», для додаткового розділу та всіх розділів, за винятком системного, є видалення.

Ініціалізація графіку повинна відбуватися кожного разу при зміні структури дисків і розділів. При ініціалізації стирається старий графік і

створюється новий, що відображає актуальну інформацію про диски і розділи. Код ініціалізації графіка представлений нижче.

```
    /// <summary>
    /// Создание графика по новым данным о томах
    /// </summary>
    private void InitGraph()
    {
        //если сплиттер в дефолтном состоянии, устанавливаем видимую часть графика для одного или
более дисков
    if (_disksList.Length <= 1 &&
    (mainSplit.SplitterDistance == 330 || mainSplit.SplitterDistance == 270))
    {
            mainSplit.SplitterDistance = 330;
            }
    else if (mainSplit.SplitterDistance == 330 || mainSplit.SplitterDistance == 270)
        {
            mainSplit.SplitterDistance = 273;
        }
        try
        {
            if (_graph != null)
            {
                _graph.Dispose();
            }
            _graph = new DiskGraphControl(scrGraph,
            _disksList) //инициализация графика
            {
                //Size = new Size(100, 100),
                Anchors = new Native.RECT(0, 0, 0, 0),
                Visible = true,
            };
            _graph.SelectedPartitionChanged +=
            _graph_SelectedPartitionChanged;
        }
        catch (Exception)
        {
            MessageBox.Show(this, "Ошибка создания графика", "Ошибка", MessageBoxButtons.OK,
MessageBoxIcon.Error);
        }
    }
```

Реалізація більшості функцій головного меню здійснюється за допомогою виклику відповідних форм, що дозволяють користувачеві задати параметри кожної дії і викликають відповідну функцію бібліотеки [9]. Прикладом таких дій можуть служити функції зміни букви розділу і форматування розділу, код яких представлений нижче.

```
    /// <summary>
    /// Изменить букву выбранного раздела
    /// </summary>
    private void ChangeLetter()
    {
        var tli = diskList.SelectedItem as TreeListItem;
    if (tli != null)
        {
            var tag = (Partition)tli.Tag;
            ChangeLetterForm chlf = new
            ChangeLetterForm(this, tag);
            chlf.ShowDialog();
            UpdateInformation();
        }
    }
```

```
/// <summary>
/// Форматировать выбранный раздел
/// </summary>
private void FormatPartition()
{
    var tli = diskList.SelectedItem as TreeListItem;
    if (tli != null)
    {
        var tag = (Partition)tli.Tag;
        var formatForm = new FormatForm(this, tag);
        formatForm.ShowDialog();
        UpdateInformation();
    }
}
```

Слід вважати, що після виконання відповідних форм додатку, структура розділів і дисків змінюється, тому в кінці виконання кожної функції викликається метод UpdateInformation (), що викликає в собі перезаповнення списку дисків і розділів, а також перебудовує графік.

5.3 Реалізація бібліотеки для роботи з файловою системою

Розробка бібліотеки для роботи з файловою системою здійснювалася за принципом ООП. Стояло завдання переробити взаємодію з функціями некерованої бібліотеки для роботи з файловою системою таким чином, щоб їх функціонал укладався в об'єктно-орієнтовану модель. Було вирішено використовувати два класи Disk, Partition, які являють собою моделі диску і розділу відповідно. Найважливішим полем даних класів є поле, що містить описувач диску або розділу. Ці описувачі відкривалися при створенні нового об'єкта заданого типу і закривалися при знищенні об'єкта. Отримання списку диску було реалізовано через статичний метод Enumerate(), отримання масиву розділів у кожного диску реалізовано через властивість m_Partitions. Решта функцій нативної бібліотеки викликалися з передачею в якості параметру описувача диску або розділу.

5.4 Реалізація графіку

Графік дисків і розділів реалізує графічне представлення файлової системи. Кінцевий вигляд заповненого графіку представлений на Рис 7.

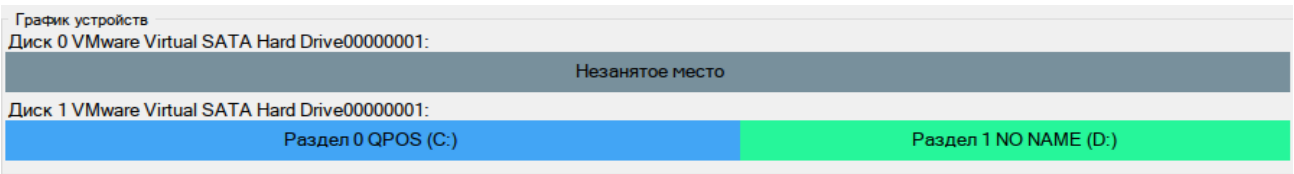

Рисунок 7 - Графік пристроїв

На графіку в рядок вказані імена всіх доступних дисків системи і рисується смуга розділів, на якій кожен колір відповідає одному з розділів, а розмір, займаний кольором, відповідає займаному розділом місця на диску щодо повного розміру диску в відсотковому співвідношенні. Код методу відтворення графіку представлений нижче.

```
protected override void OnPaint(PaintEventArgs e)
  {
      if (_isDrawingError)
          return;
      _positions.Clear();
      _curColor = Color.FromArgb(0x42, 0xa5, 0xf5);
      //задается стартовый цвет
      int x = 0;
      int y = 0; //координаты для отрисовки первой части
      графика
      int xToStr = 8;
      int yToStr = 40; //координаты для отрисовки подписи
      int count = 0;
      try
      {
          for (int i = 0; i < _disksArr.Length; i++)
          {
              _curColor = Color.FromArgb(0x42, 0xa5, 0xf5); //задается стартовый цвет
              _b = new SolidBrush(_curColor);
              e.Graphics.DrawString(_disksArr[i].m_DiskName + ":", _strFont,
              _strBrush, 0, y);
              y += 16;
              foreach (var drive in
              _disksArr[i].m_Partitions) //рисует график для каждого раздела в коллекции
          {
                  float drivePart = (float)
                  drive.ui64FullSize * 100 / _fullDrivesSize[i]; //вычисляется значение
                  занимаемого разделом места в процентах
                  int width = Convert.ToInt32((decimal)
                  drivePart * (decimal)_oneSizePart); //вычисляется ширина прямоугольника в
графике
          if (drive.PartType != "Неразмеченная
          область" && drive.PartType != "Расширенный раздел")
          {
                      e.Graphics.FillRectangle(_b, x, y, width, 30); //рисуется прямоугольник
                  }
              else
                  {
                      e.Graphics.FillRectangle(_freeSpaceBrush, x, y, width, 30);
                  }
                  if (drive.PartName.Length * 8 < width)
                  {
                      e.Graphics.DrawString(drive.PartName, _strFont, _strBrush, x +
                      (width / 2) - drive.PartName.Length * 7 / 2, y + 6); //подписывается каждый
раздел, если хватает места
                  }
                  _positions.Add(new PositionOnGraph(x, y, x + width, y + 30, drive));
          if (drive == _highlightPartition)
                  {
                      e.Graphics.DrawRectangle(_p, x,
                      y, width - 1, 30);
                  }
                  x += width; //увеличивает координату на длину отрисованного прямоугольника
          if (DrawLegend)
                  {
                      string sLetter =
```

```
                        drive.Letter.ToString();
                        if (sLetter == "\0")
                        {
                            sLetter = FREE_SPACE;
                        }
                        else
                        {
                            sLetter += ":";
                        }
                        e.Graphics.DrawString(sLetter +
                        " " + SizeFormatter.FormatDiskSize((ulong)drive.ui64FullSize), _strFont,
_strBrush, xToStr, yToStr); //подписывается имя диска и размер
                        e.Graphics.FillRectangle(_b,
                        xToStr + 220, yToStr + 3, 12, 12);
                        //подрисовывается прямоугольник, показывающий цвет диска в графике
                        count++;
                        yToStr += 18;
                        if (count == 4)
                        {
                            xToStr += 250;
                            yToStr = 40;
                            count = 0;
                        }
                    }
                    _curColor = GetNextColor(_curColor);
                    //устанавливается следующий цвет
                    _b.Dispose();
                    _b = new SolidBrush(_curColor);
                    //обновляется кисть
                }
                x = 0;
                y += 35;
            }
        }
        catch
        {
            _isDrawingError = true;
        }
    }
```

Для рисування графічних елементів на формі програми використовуються ресурси рисування з простору імен System.Painting. Дані функції дозволяють нарисувати будь-який з графічних примітивів, наприклад, прямокутник, лінію або рядок [10]. Рисування всього графіку здійснюється за допомогою проходу в циклі кожного диску і проходу кожного розділу у вкладеному циклі.

5.5 Реалізація форми форматування

Основною складністю при реалізації форми форматування є визначення доступних для розділу файлових систем. Їх список виходить з усіх підтримуваних операційною системою варіантів, крім тих файлових систем, які не підходять до заданої користувачем розділу. Отримання списку файлових систем здійснюється за допомогою функції GetFileSystemList() класу Partition, яка формує список, виходячи з особливостей конкретного розділу. Оскільки форматування розділу - це тривала операція, виклик функції форматування повинен викликатися з окремої нитки для того, щоб уникнути блокування

користувальницького інтерфейсу під час форматування. У цей момент на формі відображається індикатор активності, щоб повідомити користувача про те, що виконується процес форматування. Після повернення управління методом форматування з іншої нитки закривається форма форматування за допомогою методу Control.Invoke(). Форма форматування зображена на Рис 8.

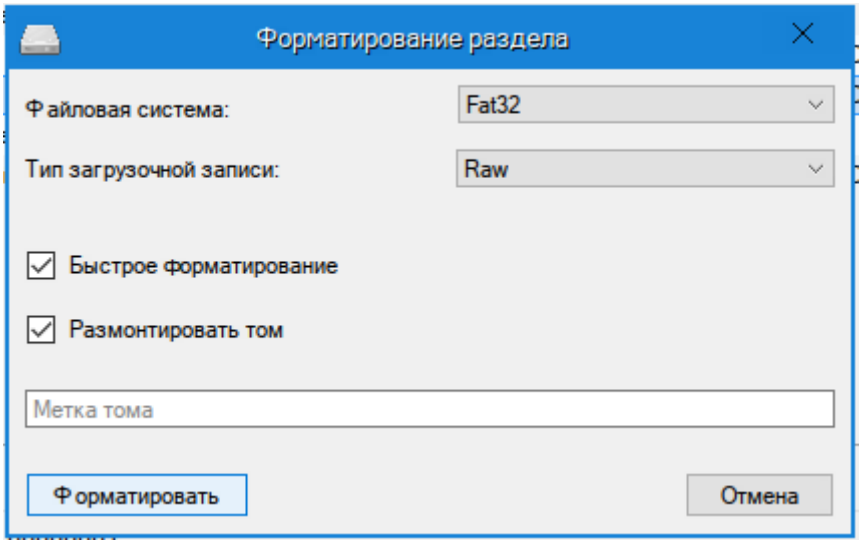

Рисунок 8 - Форма форматування

На формі форматування користувачеві представлені основні параметри форматування. За допомогою компоненти ComboBox користувачеві пропонується вибрати файлову систему і тип завантажувального запису [10]. За допомогою компоненти CheckBox користувач може вказати, чи потрібно йому швидке форматування і демонтуються томи.

5.6 Реалізація форми створення розділу

За допомогою форми створення розділу користувач повинен мати можливість вказати розмір створюваного розділу, а також, чи повинен бути створюваний розділ фізичним або ж розширеним. При цьому потрібно враховувати, що при типі завантажувального запису MBR може бути тільки 4 фізичних розділи. Розширений розділ може бути всього лише один. Дані обмеження перевіряються при відкритті форми створення розділу, і на формі відображається тільки той набір параметрів, який користувач може змінити.

6 Тестування програми

6.1 Опис розробленої програми

Для коректного тестування кінцевого додатка в першу чергу необхідно визначити можливі варіанти використання програми, дії користувача при різних сценаріях і на основі отриманої інформації скласти набір необхідних для проходження тестів. За допомогою діаграми варіантів використання складений набір тестів, представлений на таблиці 8.

Таблиця 8 - Тести додатки

Тест	Дії	Успішність виконання
1. Додати розділ жорсткого диска.	У списку дисків і розділів вибрати «Незайняте місце», у вікні натиснути «Створити».	Новий розділ створено, тест виконаний.
2. Видалити розділ жорсткого диска.	У списку дисків і розділів вибрати розділ, вибрати пункт меню «Видалити» і в меню підтвердити видалення.	Розділ видалений, тест виконаний.
3. Форматувати розділ.	У списку дисків і розділів вибрати розділ, вибрати пункт меню «Форматувати», у вікні задати параметри форматування.	Форматування завершено успішно, активна форма повідомляла про процес форматування, тест виконаний.

Продовження таблиці 8

Тест	Дії	Успішність виконання
4. Змінити активність розділу.	У списку дисків і розділів вибрати розділ, вибрати пункт меню «Зробити активним ».	Розділ став активним, прапор активності з попереднього розділу знятий, тест виконаний.
5. Змінити тип завантажувального запису	У списку дисків і розділів вибрати диск, вибрати пункт меню «Перетворити в GPT», підтвердити перетворення.	Вся інформація з диску видалена, тип завантажувального запису диску змінено, тест виконаний.

Крім основних варіантів використання було змодельовано кілька нестандартних варіантів взаємодії з системою. У цих випадках перевірялася коректна реакція програми на помилкові ситуації і відмовостійкість системи. Список тестів з варіантами нестандартного або помилкового використання додатку представлений в таблиці 9. На Рис 9 - 11 зображені результати запропонованих тестів.

Таблиця 9 - Тести некоректного використання додатка

Тест	Дії	Успішність виконання
1. Запуск додатку з нестачею прав	Запустити програму користувачем, у якого немає прав на роботу з дисками	Додаток виконав перевірку на наявність необхідних прав при запуску, повідомило користувачу про їх нестачі. Тест виконаний.

Продовження таблиці 9

Тест	Дії	Успішність виконання
2. Запуск другої копії додатку	Запустити копію додатку, коли одна копія вже запущена.	Додаток не зміг отримати список дисків, Оскільки їх описувачі вже відкриті, видало відповідне повідомлення.
3. Спроба форматування або видалення системного розділу	У списку дисків і розділів вибрати системний розділ, вибрати пункт меню «Форматувати» або "Видалити".	Можливість форматування або видалення розділу виявилася недоступна, тест виконаний.

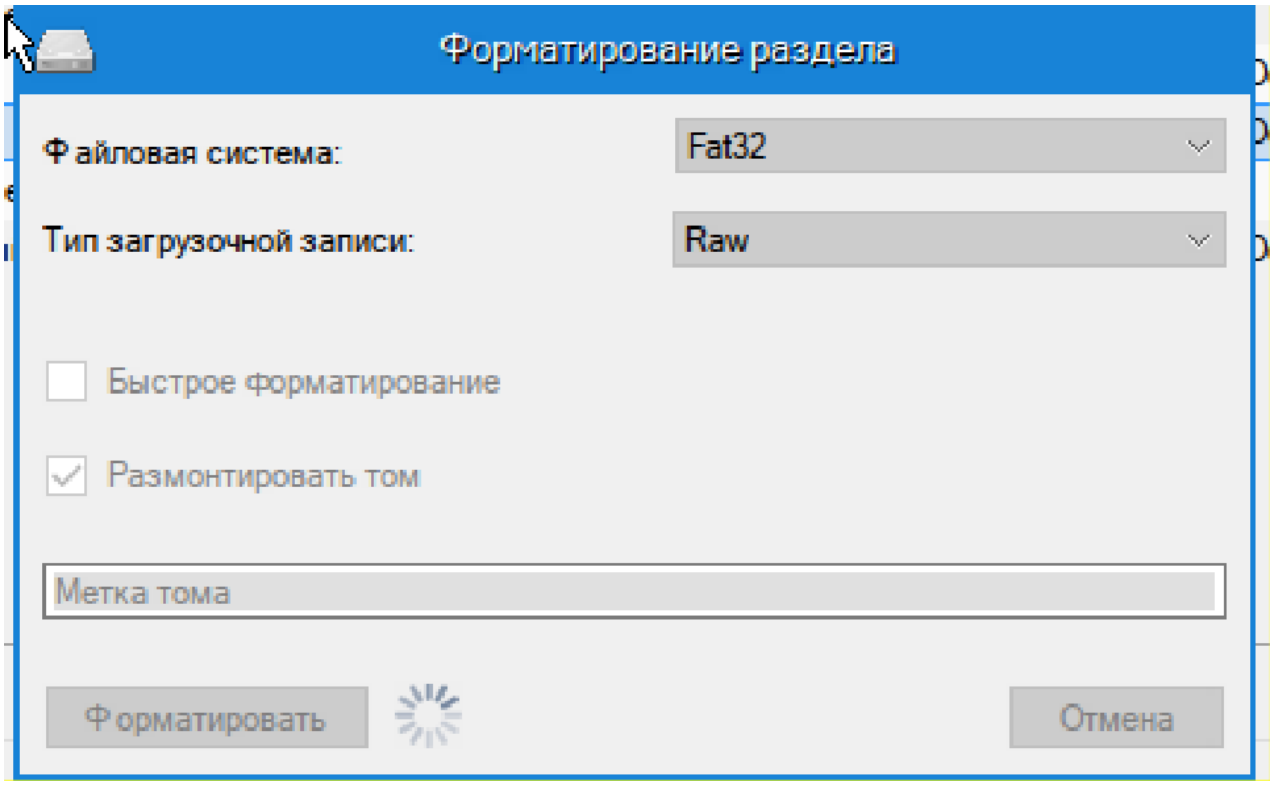

Рисунок 9 - Тестування процесу форматування

Як видно зі скріншоту, інтерфейс форми форматування прийняв вигляд очікування завершення операції. Форма забороняє користувачеві своє закриття.

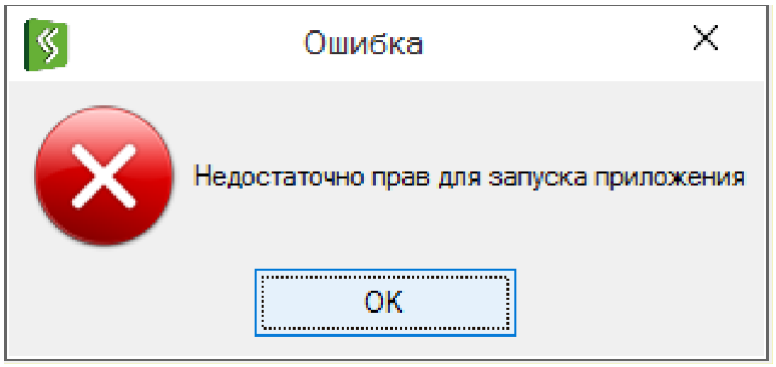

Рисунок 10 - Запуск програми з нестачею прав

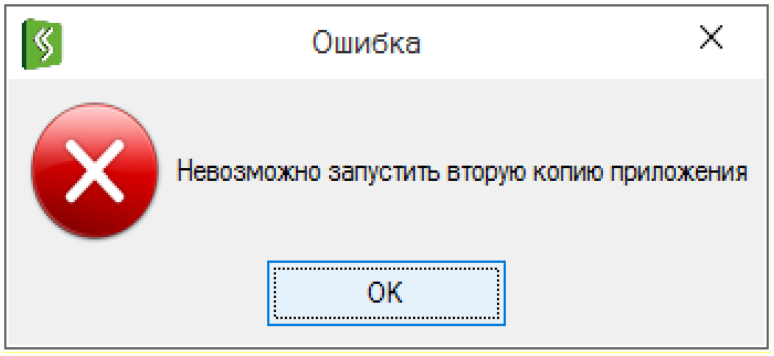

Рисунок 11 - Запуск другої копії додатку

Таким чином, на стадії тестування було встановлено, що додаток проходить всі тести, спрямовані на перевірку варіантів використання, а також тести з нестандартним використанням самої програми і її функцій. Можна зробити висновок про те, що програма готова до безпосереднього використання.

6.2 Аналіз якості програмних засобів

Метрики програмного забезпечення служать для подання будь-якої характеристики програмного забезпечення в числовому вираженні. Зазвичай підхід розрахунку метрик використовується для кількісної оцінки якості ПЗ. При визначенні вимог до програмного продукту часто задаються різні характеристики програми, що визначають різні боки управління продуктом в заданому середовищі. Для кожної характеристики якості визначається відповідна метрика, задається спосіб її визначення, а також діапазон значень.

Такий підхід до оцінки якості програмного забезпечення іноді критикують, оскільки вважається, що не можна виробити єдиний спосіб оцінки якості програми і програмного коду, який можна застосовувати для будь-якого програмного продукту. Також вважається, що не можна судити про якість коду та ефективність програміста по метриках.

Розрахунок метрик програмного забезпечення є одним з небагатьох визнаних методів оцінки якості програмного продукту. Найбільш часто використовуваними метриками є: індекс зручності підтримки, цикломатична складність, глибина успадкування, взаємозалежність класів, кількість рядків коду.

Цикломатична складність показує структурну складність коду, тобто кількість різних операційних гілок в коді. Чим не більше цей показник, тим складніше буде охопити тестами весь програмний код, тим самим може знизитися його якість. Переважно, щоб при розробці алгоритму для вирішення конкретного завдання, обирався варіант з найбільш простою структурою реалізації, оскільки такий підхід зменшує шанс виникнення непередбачених помилок. При обчисленні цикломатичної складності зазвичай використовується граф потоку керування програми. Вузли графа відповідають неподільним групам команд програми. Вузли з'єднуються ребрами, якщо одна група команд може бути виконана безпосередньо після іншої групи. Такий підхід обчислення метрики можна використовувати для окремих функцій, модулів або класів в програмі.

Глибина успадкування розраховується для одного класу програми. Дана характеристика показує, який він за рахунком в ланцюжку спадкування. Для підрахунку даної метрики для всього програмного продукту зазвичай використовується середнє значення цих показників для кожного класу програми. Менше значення даної метрики є кращим, оскільки складна ієрархія класів і численне спадкування негативно впливає на зручність підтримки програми і на читаність самого коду. Для мови C# бажано не підіймати даний показник вище 9, оскільки підрахунок ведеться від самого базового класу object.

Взаємозалежність класів показує ступінь залежності класів один з іншим. До аналізу беруться всі можливі характеристики класу і його вмісту, такі як

приватні та публічні поля, локальні змінні, аргументи функцій і їх значення, що повертається. Гарний дизайн програмного коду передбачає мала кількість зв'язків між класами. Чим їх більше, тим складніше надалі використовувати цей клас при вирішенні іншої подібної задачі, а також ускладнюється підтримка і розуміння програмного коду.

Метрика рядків коду використовується для вимірювання його обсягу за допомогою підрахунку кількості рядків в тесті вихідного коду. Зазвичай даний показник побічно показує кількість трудовитрат на реалізацію конкретного завдання і на написання конкретного програмного продукту. Традиційно вважається, що має сенс порівнювати дані метрики лише з точністю до порядку, оскільки цей показник не є достатньо достовірним і може відрізнятися в залежності від стилю написання коду і від мови програмування.

Для кожної мови використовуються свої алгоритми розрахунку рядків коду для збільшення об'єктивності даної метрики.

6.3 Кількісні метрики

В першу чергу слід розглянути кількісні характеристики вихідного коду програм з огляду на їх простоту. Найелементарнішою метрикою є кількість рядків коду (SLOC). Дана метрика була спочатку розроблена для оцінки трудовитрат за проєктом. Таким же чином до групи метрик, заснованих на підрахунку деяких одиниць в коді програми, відносять метрики Холстеда. Дані метрики засновані на наступних показниках:

η_1 - число унікальних операторів програми, включаючи символи- роздільники, імена процедур і знаки операцій (словник операторів);

η_2 - число унікальних операндів програми (словник операндів);

N_1 - загальне число операторів в програмі;

N_2 - загальне число операндів в програмі.

З огляду на введені позначення, можна визначити: $\eta = \eta_1 + \eta_2$ -

словник програми;

$N = N_1 + N_2$ - довжина програми;

$HV = Nlog_2\eta$ - обсяг програми.

При застосуванні метрик Холстеда частково компенсуються недоліки, які пов'язані з можливістю запису однієї і тієї ж функціональності різною кількістю

операторів і рядків.

Всі перераховані вище розрахунки проведемо для методу «UpdateInformation» класу MainForm, який здійснює відрисовку графіку дисків і розділів. Лістинг даного методу представлений нижче.

```
private void UpdateInformation()
    {
        _isUpdate = true;
        try
        {
        //закрытие каждого диска перед тем как открыть их заново
        foreach (var item in diskList.Items.Items())
            {
                if (item.Tag is Disk)
                {
                    var disk = (Disk)item.Tag;
                    disk.Dispose();
                }
            }
            //очистка дерева и списков для перезаполнения
            diskList.Items.Clear();
            //UpdateNewInformation();
            if (_sCurrPath != null)
            {
                diskList.OpenItem(_sCurrPath);
            }
            InitGraph();
        }
        catch
        {
            MessageBox.Show(this, "Ошибка обновления списка дисков и разделов", "Ошибка",
MessageBoxButtons.OK, MessageBoxIcon.Error, MessageBoxButton.Ignore);
        }
        _isUpdate = false;
```

}підрахуємо показники η_1, η_2, N_1, N_2.

Результати підрахунків наведені у таблиці 10.

Таблиця 10 - Розрахунок основних показників для методу UpdateInformation

Оператори		Операнди	
назва	кількість	назва	кількість
;	5	_partitionsList	1
=	4	_disk	2
If	2	_isUpdate	1
=3	N_1=11	=3	N_2=4

Звідси отримуємо словник програми $\eta = \eta_1 + \eta_2 = 6$ і довжину програми $N = N_1 + N_2 = 15$. Отже, можна вирахувати обсяг програми по формулі

$HV_1 = N log_2 \eta = 15 log_2 6 = 38{,}775$.

6.4 Індекс зручності підтримки

MI = MAX (0, (171 - 5.2 * ln (HV) - 0.23 * CC - 16.2 * ln (LoC)) * 100/171), де HV - Halstead Volume, обчислювальна складність. Чим більше операторів, тим більше значення цієї метрики;

CC – цикломатична складність;

LoC - кількість рядків коду.

Ця метрика може приймати значення від 0 до 100 і показує відносну складність підтримки коду. Чим більше значення цієї метрики, тим легше підтримувати код.

6.5 Глибина успадкування

Для розрахунку метрики глибини успадкування використовуємо класи додатки MainForm і DiskGraphControl додатки.

Для класу MainForm ієрархія успадкування буде представлена в наступному вигляді: Object-Control: IDisposable-Form-MainForm. Таким чином, метрика «глибина спадкування» для цього класу буде дорівнює 4.

Клас DiskGraphControl успадкований від класу Scrollable, який в свою чергу є дочірнім класом класу Control. За аналогією з попереднім класом, ієрархія успадкування буде виглядати наступним чином: Object-Control: IDisposable-Scrollable-DiskGraphControl. Таким чином, метрика
«Глибина спадкування» для класу DiskGraphControl буде дорівнює 5.

Оскільки середнє значення метрики глибини успадкування дорівнюватиме 4.5, можна сказати, що цей додаток задовольняє вимогам щодо глибини успадкування.

6.6 Загальний розрахунок показників програмного продукту

Для розрахунку інших показників скористаємося стандартними можливостями середовища розробки Microsoft Visual Studio. Всі метрики програмного забезпечення відображені в таблиці 5.

Таблиця 10 - Значення метрик

Метрика	Значення
індекс зручності підтримки	75
цикломатична складність	554
глибина успадкування	4
взаємозалежність класів	109
рядки коду	1949

Як випливає з таблиці, метрики програмного коду відповідають допустимим нормам для програмного продукту. Із значення індексу зручності підтримки можна зробити висновок, що подальша підтримка програми не буде ускладнена зайвою алгоритмічною складністю і додатковими залежностями між класами.

7 Опис та інструкція по використанню програми

Розроблений додаток - утиліта для роботи з дисками і розділами, доступними операційній системі. Програма дозволяє виконувати основні дії з розміткою диска, типом завантажувального запису, типами файлової системи розділів і т.д.

Для функціонування програми розроблена бібліотека для роботи з низькорівневими функціями операційної системи. Бібліотека розроблена в рамках ООП і не має прямих залежностей від графічної частини програми, що дозволяє використовувати розроблену бібліотеку в інших програмних

продуктах, в яких потрібна робота з файловою системою.

Також в додатку розроблений графік дисків і розділів, який реалізує графічне представлення дисків і розділів в операційній системі. Графік дозволяє наочно оцінити обсяг пам'яті, який використовується будь-якими розділами на диску, а також незайнятий обсяг пам'яті, який можна використовувати при створенні нових розділів. Крім інтерактивної складової графік є інтерактивним, що дозволяє виділяти на графіку потрібний диск або розділ для подальших операцій над ним.

У додатку реалізовані форми властивостей для дисків і розділів, що дозволяє користувачеві оцінити їх основні характеристики.

Додаток має сучасний графічний інтерфейс. Графічний інтерфейс розробленого додатка відрізняється швидкістю роботи, можливістю зміни розмірів форми, а також створений згідно керівних документів для розробки додатків. Пропорції вікон для відображення інформації не фіксовані, а мають можливість динамічної зміни. Таку можливість має елемент управління SplitterContailner, який використовувався для позиціювання. У додатку використовується механізм багатопоточності для виконання операцій, що вимагають багато часу для завершення. Використання окремої нитки для виконання таких операцій дозволяє уникнути блокування призначеного для користувача інтерфейсу, яка могла б відбуватися до тих пір, поки тривало виконуюча функція не повернула б управління викликаючому коду в GUI-нитці. Взаємодія ниток з ниткою призначеного для користувача інтерфейсу відбувається через механізм Invoke. Для організації багатопоточності використовувався механізм ThreadPool. В його основі робота з набором ниток, які розподіляються між завданнями. У разі нестачі ниток створюються нові, якщо нитки простоюють, їх кількість скорочується. Завантаженість ниток підтримується на постійному рівні, тримаючи баланс між швидкістю виконання завдання і відсотком незавантажених ниток.

Після запуску встановлена програма відкриває головне вікно. На даному вікні представлений деревоподібний список всіх дисків і розділів. У верхній частині вікна розташовано меню, на якому представлені операції, доступні для обраного в цей час диску або розділу. При виборі кожного з пунктів меню

викликається відповідна форма, що дозволяє користувачеві задати необхідні параметри виконання запропонованої дії. Для багатьох функцій вас запитають, оскільки при роботі з файловою системою є великий ризик втрати даних. У нижній частині відображається графік, що відображає поточний стан файлової системи. Графік є інтерактивним, при натисканні на будь-який з його елементів аналогічний елемент виділяється в основному списку. Також для будь-якого диску і розділу є відображення властивостей.

Для форматування розділу необхідно вибрати потрібний розділ в списку або на графіку і натиснути пункт меню «Форматувати». На запропонованій формі потрібно вказати файлову систему і тип форматування. Потім необхідно підтвердити виконання дії і дочекатися закінчення форматування.

Для створення нового розділу на жорсткому диску необхідно виділити в списку дисків або розділів одне з незайнятих місць і вибрати пункт меню «Створити розділ». У запропонованій формі потрібно вказати розмір створюваного розділу, обмежений розміром обраного незайнятого простору. Потім необхідно підтвердити виконувану дію натисканням кнопки «Створити». Для подальшого використання створеного розділу необхідно виконати для нього форматування, оскільки спочатку розділ створюється без файлової системи.

Для форматування розділу необхідно вибрати потрібний розділ в списку або на графіку і натиснути пункт меню «Видалити», потім підтвердити свою дію. При видаленні розділу всі дані на ньому будуть безповоротно втрачені. Розмір віддаленого розділу при видаленні буде додано до наявного незайнятому простору на диску.

Висновок

В результаті проведеної роботи була розроблена програма для роботи з розміткою жорсткого диска.

Додаток дозволяє такі дії над розміткою як створення, видалення розділу, установка прапору активності розділу, форматування. Для дисків можлива зміна типу завантажувального запису з GPT на MBR і навпаки.

Реалізована можливість перегляду властивостей дисків і розділів.

Розроблена бібліотека для роботи з некерованими функціями операційної системи для роботи з розміткою.

Також спроєктований і розроблений графік файлової системи, що відображає користувачеві наочне уявлення про диски і розділи в операційній системі.

Таким чином, вимоги завдання виконані в повному обсязі.

Розроблений додаток допоможе у вивченні протоколу принципів роботи з розміткою жорстких дисків, а якісна реалізація дозволяє використовувати додаток в повсякденному житті в якості утиліти для роботи з файловою системою. Відмінними рисами розробленого додатка є простота і інформативність представлення інформації.

В ході виконання дипломного проєктування були продемонстровані навички, отримані в ході навчання в СумДУ. Такі як: вміння програмувати об'єктно-орієнтованою мовою програмування, вміння тестувати програмні продукти, вміння розбиратися в специфікаціях, а також вміння писати документацію.

Список використаних джерел

1. Вікіпедія.Файловая система. [Електронний ресурс]. URL: https://ru.wikipedia.org/wiki/Файловая_система
2. Вікіпедія. Головний завантажувальний запис. [Електронний ресурс]. URL: https://ru.wikipedia.org/wiki/Главная_загрузочная_запись
3. Вікіпедія. Таблиця розділів GUID. [Електронний ресурс]. URL: https://ru.wikipedia.org/wiki/ Таблица_разделов_GUID
4. Зіборов, В. VisualC # 2010 на прикладах. / В. Зіборов - СПб. БХВ-Петербург, 2011. - 480 с
5. Арло, Д. UML 2 і Уніфікований процес. Практичний об'єктно орієнтований аналіз і проєктування / Д. Арло, А. Нейштадт - М: Видавництво «Символ-Плюс», 2007 - 624 с.
6. Уотсон, К. Visual C # 2010: повний курс. Пер. з англ. / К. Уотсон, К. Нейгел, Я. Педерсен, Д. Д. Рід, М. Скіннер. - М .: Діалектика, 2010. - 960 с.
7. Вікіпедія. Windows Forms. [Електронний ресурс]. URL: https://ru.wikipedia.org/wiki/Windows_Forms (дата звернення 22.05.2017).
8. Види тестування програмного забезпечення. [Електронний ресурс]. URL:http://fb.ru/article/93418/vidyi-testirovaniya-programmnogo-obespecheniya
9. Електронна бібліотека [Електронний ресурс]. URL: http://msdn.microsoft.com
10. Метрики коду програмного забезпечення / PVS-Studio: сайт. 2017. [Електронний ресурс]. URL: https: //www.viva64.com/ru/a/0045/
11. Вікіпедія. Java. [Електронний ресурс]. URL: https://uk.wikipedia.org/wiki/Java
12. Вікіпедія. Pascal. [Електронний ресурс]. URL: https://uk.wikipedia.org/wiki/Pascal
13. Вікіпедія. C Sharp. [Електронний ресурс]. URL: https://uk.wikipedia.org/wiki/C_Sharp

Додаток А

ГЛОСАРІЙ

Файлова система - порядок, що визначає спосіб організації, зберігання та іменування даних на носіях інформації в комп'ютерах, а також в іншому електронному обладнанні: цифрових фотоапаратах, мобільних телефонах та іншому. Файлова система визначає формат вмісту і спосіб фізичного зберігання інформації, яку вважається групувати у вигляді файлів. Конкретна файлова система визначає розмір назв файлів та (каталогів), максимальний можливий розмір файлу і розділу, набір атрибутів файлу. Деякі файлові системи надають сервісні можливості, наприклад, розмежування доступу або шифрування файлів.

Жорсткий диск - пристрій (пристрій зберігання інформації) довільного доступу, заснований на принципі магнітного запису. Є основним накопичувачем даних в більшості комп'ютерів.

Операційна система - комплекс взаємопов'язаних програм, призначених для управління ресурсами комп'ютера та організації взаємодії з користувачем. У логічній структурі типової обчислювальної системи операційна система займає положення між пристроями з їх мікроархітектурою, машинною мовою та можливо, власними (вбудованими) мікропрограмами (драйверами) з одного боку і прикладними програмами з іншого.

Головний завантажувальний запис, код і дані необхідні для подальшого завантаження операційної системи і розташовані в перших фізичних секторах (найчастіше в найпершому) на жорсткому диску або іншому пристрої для збереження інформації.

MBR містить невеликий фрагмент коду, що виконується, таблицю розділів (partition table) і спеціальну сигнатуру. Функція MBR - «перехід» в той розділ жорсткого диска, з якого слід виконувати «подальший код» (зазвичай - завантажувати ОС). На «стадії MBR» відбувається вибір розділу диску, завантаження коду ОС (відбувається на більш пізніх етапах алгоритму).

NTFS - стандартна файлова система для сімейства операційних систем Windows NT фірми Microsoft. NTFS підтримує зберігання метаданих. З метою поліпшення продуктивності, надійності і ефективності використання дискового простору для зберігання інформації про файли в NTFS використовуються спеціалізовані структури даних.

Додаток Б

Лістинг

```
public partial class MainForm : Form
{
   private DiskGraphControl _graph; //график
   private readonly List<Partition> _partitionsList;
   //список разделов
   private Disk[] _disksList;
   private string _sCurrPath;
   private readonly string _sSystemPath;
   private const string FREE_SPACE = "Незанятое место";
   private const string EXT_PARTITION = "Расширенный раздел";
   private const string LOGICAL_PARTITION = "Логический раздел";
   private const string FILE_SYSTEM_FAT32 = "FAT32";
   private const string FILE_SYSTEM_FREE = "Неразмеченная область";
   private bool _isUpdate;
   public MainForm()
   : base(null)
   {
      InitializeComponent();
      Activate();
      this.CenterAt(CenterOrigin.WorkingArea);
      _partitionsList = new List<Partition>();
      HideTools();
      if (Application.IsQpOs) //заполнение списка дисков и томов
   {
         var pBuf = new StringBuilder(50);
         SplitterDllInterface.OsGetFullSystemDirectory(pBuf, 50);
         _sSystemPath = pBuf.ToString();
         //try
         //{
         //UpdateNewInformation();
         FillDisksInfo();
         //}
         //catch (Exception)
         //{
         // MessageBox.Show(this, "Ошибка получения информации о дисках", "Ошибка",
MessageBoxButtons.OK,MessageBoxIcon.Error);
         //}
   InitGraph();
      }
      this.VScroll.Dispose();
      this.HScroll.Dispose();
      scrGraph.HScroll.Dispose();
      diskList.CreateSubMenuFunc = (Control target) =>
      //Меню для каждого элемента в списке дисков
      {
         if (target.Tag is Disk)
         {
            var menu = new PopupMenuStrip(diskList);
            var tag = target.Tag as Disk;
            menu.AddItem("Информация о диске",
            Disk_Info); //информация получается из любого диска
            if (tag.PartStyle ==
            PARTITION_STYLE.PARTITION_STYLE_GPT) //для структруры gpt диск можно преобразовать в mbr
         {
               menu.AddItem("Преобразовать в MBR диск\tAlt + M", Menu_ToMBR);
         }
else if (tag.PartStyle ==
```

```
PARTITION_STYLE.PARTITION_STYLE_MBR) //для mbr наоборот
            {
               menu.AddItem("Преобразовать в GPT диск\tAlt + G", Menu_ToGPT);
               menu.AddItem("Записать загрузочный код MBR\tAlt + B", Menu_WriteCode);
}
            else if (tag.PartStyle ==
            PARTITION_STYLE.PARTITION_STYLE_RAW)
            {
               menu.AddItem("Преобразовать в MBR диск\tAlt + M", Menu_ToMBR);
               menu.AddItem("Преобразовать в GPT диск\tAlt + G", Menu_ToGPT);
            }
            return null;
         }
         else if (target.Tag is Partition && target.Text
         != FREE_SPACE)
            //если выбран раздел, не являющийся свободным пространством
      {
      var tag = (Partition)target.Tag;
            //получаем информацию о выбранном разделе
            var menu = new PopupMenuStrip(diskList);
            if (!tag.Active) //если раздел не является активным, его можно сделать активным
      {
               menu.AddItem("Сделать активным\tAlt + A", Menu_ToActive);
      }
else
            {
               menu.AddItem("Убрать активность\tAlt + A", Menu_ToUnactive);
}
            //у любого раздела (кроме системного) можно изменить букву, метку и загрузочный код
      menu.AddItem("Изменить метку\tAlt+W",
      Menu_ChangeMark);
            menu.AddItem("Изменить загрузочный код раздела\tAlt + P", Menu_WriteLoadBootCode);
      if
      (!_sSystemPath.StartsWith(tag.Letter.ToString()) || tag.Letter ==
      '\0')
               //если раздел не системный или у него нет буквы, его можно удалить, првоерить, форматировать
      {
               menu.AddItem("Изменить букву\tAlt+L", Menu_ChangeLetter);
               menu.AddItem("Удалить раздел\tDelete", Menu_Delete);
               menu.AddItem("Проверить раздел\tAlt + C", Menu_CheckPartition);
               menu.AddItem("Форматировать\tAlt+F", Menu_Format);
               if (tag.FileSystemName ==
               FILE_SYSTEM_FAT32) //если файловая система fat32, раздел можно перевести в splitfat
      {
                  menu.AddItem("Конвертировать в SplitFat", Menu_ToSplitFat);
      }
            }
            if
            (!_sSystemPath.StartsWith(tag.Letter.ToString()) && tag.Letter !=
            '\0')
      {
               menu.AddItem("Удалить букву", Menu_DeleteLetter);
            }
            menu.AddItem("Свойства раздела\tAlt+T", Menu_PartitionProperties);
            if (menu.Controls.Count > 0) //чтобы не отображать меню без элементов
      return menu;
            return null;
         }
else if (target.Text == FREE_SPACE) //если выбранный раздел является свободным пространством
      {
            var vi = (Partition)target.Tag;
            //получение информации о разделе
            var menu = new PopupMenuStrip(diskList);
            if (vi.PartType == EXT_PARTITION) //если раздел расширенный
      {
               bool isLogic = false;
               //проверяем разделы на наличие расширенного
```

```
        foreach (var volume in
        _partitionsList)
                {
                   if (volume.sDisk.Handle ==
                   vi.sDisk.Handle)
                   {
                      if (volume.PartType ==
                      LOGICAL_PARTITION)
                      {
                         isLogic = true;
                      }
                   }
                }
                //если раздел расширенный и отстутствуют логические, разедел можно удалить
        if (vi.PartType == EXT_PARTITION)
                {
                   if (!isLogic)
                   {
                      menu.AddItem("Удалить раздел\tDelete", Menu_Delete);
                   }
        }
                menu.AddItem("Создать раздел\tInsert", Menu_CreateNew);
        }
else
             {
                int partitionCount = 0;
                foreach (var volume in
                _partitionsList)
                   //проверка на количество существующих разделов, их может быть не больше четырех
{
                   if (volume.sDisk.Handle ==
                   vi.sDisk.Handle)
                   {
                      if (volume.PartType !=
                      LOGICAL_PARTITION)
                      {
                         if (volume.PartType !=
                         FILE_SYSTEM_FREE)
                            partitionCount +=
                            1;
                      }
                   }
                }
                if (vi.sDisk.PartStyle ==
                PARTITION_STYLE.PARTITION_STYLE_MBR)
                {
                   if (partitionCount < 4)
                   {
                      menu.AddItem("Создать раздел\tInsert", Menu_CreateNew);
                   }
                }
                else
                {
                   menu.AddItem("Создать раздел\tInsert", Menu_CreateNew);
                }
             }
             if (menu.Controls.Count > 0) //чтобы не отображать меню без элементов
        return menu;
              return null;
           }
          return null;
        };
     var sorter = new Sorting();
        diskList.ItemSorter = sorter;
     }
     /// <summary>
     /// Удаление инструментов из панели
```

```
    /// </summary>
    private void HideTools()
    {
        /*
        * Прибитие анкерами чекбокса приводит к тому, что
        все добавленные после него тулитемы будут учитывать отступ
        чекбокса
        * */
        foreach (var control in MainMenuStrip.Children())
        {
            if (control != cbShowGraph)
            {
                control.Visible = false;
            }
        }
    }
    /// <summary>
    /// Отображение элементов панели инструментов в зависимости от выбранного в дереве элемента
/// </summary>
private void ShowTools()
    {
        tiRefr.Visible = true;
        //отображение элементов панели инструментов аналогично отображению пунктов меню при вызове контекстного
меню
var target = diskList.SelectedItem as TreeListItem;
        if (target == null)
            return;
        if (target.Tag is Disk)
        {
            var tag = target.Tag as Disk;
            tiProp.Visible = true;
            if (tag.PartStyle ==
            PARTITION_STYLE.PARTITION_STYLE_GPT)
            {
                tiToMbr.Visible = true;
            }
            else if (tag.PartStyle ==
            PARTITION_STYLE.PARTITION_STYLE_MBR)
            {
                tiToGpt.Visible = true;
            }
            else if (tag.PartStyle ==
            PARTITION_STYLE.PARTITION_STYLE_RAW)
                // диск с разметкой raw можно преобразовать в любую другую разметку
        {
                tiToGpt.Visible = true;
                tiToMbr.Visible = true;
            }
            //tiWriteLoadCode.Visible = true;
        }
        else if (target.Tag is Partition && target.Text != FREE_SPACE)
        {
            var vi = (Partition)target.Tag;
            if (!vi.Active)
            {
                tiMakeActive.Visible = true;
            }
            else
            {
                tiUndoActive.Visible = true;
            }
            tiChangeLabel.Visible = true;
            if
            (!_sSystemPath.StartsWith(vi.Letter.ToString()) || vi.Letter == '\0')
            {
                tiChangeLetter.Visible = true;
                tiDelete.Visible = true;
```

```
            tiFormat.Visible = true;
            if (vi.FileSystemName ==
            FILE_SYSTEM_FAT32)
            {
               //tiToSplitFat.Visible = true;
            }
         }
      }
      else if (target.Text == FREE_SPACE)
      {
         var vi = (Partition)target.Tag;
         if (vi.PartType == EXT_PARTITION)
         {
         tiCreate.Visible = true;
            bool isLogic = false;
            foreach (var volume in _partitionsList)
            {
               if (volume.sDisk.Handle ==
               vi.sDisk.Handle)
               {
                  if (volume.PartType ==
                  LOGICAL_PARTITION)
                  {
                     isLogic = true;
                  }
               }
            }
            if (vi.PartType == EXT_PARTITION)
            {
               if (!isLogic)
                  tiDelete.Visible = true;
            }
         }
         else
         {
            int partitionCount = 0;
            foreach (var volume in _partitionsList)
            {
               if (volume.sDisk.Handle ==
               vi.sDisk.Handle)
               {
                  if (volume.PartType !=
                  LOGICAL_PARTITION)
                  {
                     if (volume.PartType !=
                     FILE_SYSTEM_FREE)
                        partitionCount += 1;
                  }
               }
            }
            if (vi.sDisk.PartStyle ==
            PARTITION_STYLE.PARTITION_STYLE_MBR)
            {
               if (partitionCount < 4)
                  tiCreate.Visible = true;
            }
            else
            {
               tiCreate.Visible = true;
            }
         }
      }
   }
/// <summary>
/// Создание графика по новым данным о томах
/// </summary>
private void InitGraph()
```

```
    {
        //если сплиттер в дефолтном состоянии, устанавливаем видимую часть графика для одного или более дисков
if (_disksList.Length <= 1 &&
(mainSplit.SplitterDistance == 330 || mainSplit.SplitterDistance
== 270))
        {
            mainSplit.SplitterDistance = 330;
        }
        else if (mainSplit.SplitterDistance == 330 ||
        mainSplit.SplitterDistance == 270)
        {
            mainSplit.SplitterDistance = 273;
        }
        try
        {
            if (_graph != null)
            {
                _graph.Dispose();
            }
            _graph = new DiskGraphControl(scrGraph,
            _disksList) //инициализация графика
            {
                //Size = new Size(100, 100),
                Anchors = new Native.RECT(0, 0, 0, 0),
                Visible = true,
            };
            _graph.SelectedPartitionChanged +=
            _graph_SelectedPartitionChanged;
        }
        catch (Exception)
        {
            MessageBox.Show(this, "Ошибка создания графика", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
```