

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Інститут інформатики та радіоелектроніки,
Факультет комп'ютерних наук і технологій
(повне найменування інституту, назва факультету)

Кафедра програмних засобів
(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)
бакалавр
(ступінь вищої освіти)

на тему РОЗРОБКА ПРОГРАМИ ДЛЯ МОНИТОРИНГУ СТАНУ ЖОРСТКОГО
ДИСКУ. SOFTWARE DEVELOPMENT FOR HARD DRIVE MONITORING.

Виконав: студент(ка) 4 курсу, групи КНТ-218сп
Спеціальності 122 Комп'ютерні науки
(код і найменування спеціальності)

Освітня програма (спеціалізація)
Комп'ютерні науки

Тарасенко Б.В.

(прізвище та ініціали)

Керівник Сердюк С.М.

(прізвище та ініціали)

Рецензент Неласа Г.В.

(прізвище та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
 (повне найменування закладу вищої освіти)

Інститут, факультет ІРЕ, ФКНТ
 Кафедра програмних засобів
 Ступінь вищої освіти бакалавр
 Спеціальність 122 Комп'ютерні науки
 (код і найменування)
 Освітня програма (спеціалізація) Комп'ютерні науки
 (назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
С.О. Субботін
 “ ” _____ 2021 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

Тарасенка Богдана Вікторовича
 (прізвище, ім'я, по батькові)

1. Тема проєкту (роботи) Розробка програми для моніторингу стану жорсткого диску. Software Development for Hard Drive Monitoring.

керівник проєкту (роботи) доц., к.т.н. Сердюк Сергій Микитович,
 (прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом закладу вищої освіти від “30” березня 2021 року
 № 103

2. Строк подання студентом проєкту (роботи) _____ травня 2021 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Матеріали та методи. 3. Основні рішення щодо реалізації компонентів системи. 4. Керівництво програміста. 5. Керівництво оператора. Додатки.

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

6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-5 Основна частина	Сердюк С.М., доцент		
Нормоконтроль	Дейнега Л.Ю., ст. викладач		

7. Дата видачі завдання 28 лютого 2021 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	2–3 тижні	Розділ 1
3	Розробка архітектури програми.	4 тиждень	Розділ 2
4	Розробка програми.	5–6 тижні	Розділ 3
5	Тестування та експериментальне дослідження програми.	7 тиждень	Розділ 4
6	Оформлення пояснювальної записки та документів до неї. Нормоконтроль та рецензування	8–9 тижні	Розділ 5, Додатки
7	Захист роботи.	10 тиждень	

Студент(ка)

(підпис)

Тарасенко Б.В.

(прізвище та ініціали)

Керівник проєкту (роботи)

(підпис)

Сердюк С.М.

(прізвище та ініціали)

(підпис)

(прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра:
89 с., 2 табл., 11 рис., 2 дод., 18 джерела.

C#, WINDOWS, NETFRAMEWORK, ЖОРСТКИЙ ДИСК, СЕКТОР,
S.M.A.R.T.

Об'єкт дослідження – процес перевірки стану жорсткого диску.

Предмет дослідження – методи перевірки стану жорсткого диску з використанням технології S.M.A.R.T.

Мета роботи – розробка програми, яка дозволить переглядати значення S.M.A.R.T. атрибутів та проводити перевірку на наявність пошкоджених секторів обраного жорсткого диску (ЖД).

Матеріали, методи та технічні засоби: об'єктно-орієнтоване програмування; мова програмування C# та середовище розробки Visual Studio 2017; програмна технологія .NET Framework 4.6.1; персональний комп'ютер (ПК) з мікропроцесором Intel Pentium 1,0 ГГц та вище, під управлінням операційної системи (ОС) Windows 10.

Результати. Розроблено застосунок, який забезпечує перегляд S.M.A.R.T. атрибутів ЖД, здійснює пошук пошкоджених секторів та намагається їх виправити.

Висновки. Розроблено програмне забезпечення для автоматизації процесу моніторингу стану жорсткого диску.

Галузь використання – експлуатаційний сервіс комп'ютерних систем.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 89 pages, 11 figures, 2 tables, 2 appendixes, 18 sources.

C#, WINDOWS, NETFRAMEWORK, HARD DRIVE, SECTOR, S.M.A.R.T.

Object of study is the process of checking a hard disk status.

Subject of study are methods to check a hard disk status using S.M.A.R.T.

The purpose of the work is development of a program that allows you to view the values of S.M.A.R.T. attributes and check for bad sectors of the selected hard disk (HD).

Materials, methods and technical means: object-oriented programming; C# programming language and development tools Visual Studio 2017; .NET Framework 4.6.1 software technology; personal computer (PC) with Intel Pentium 1.0 GHz and higher microprocessor, running Windows 10 operating system (OS).

Results. An application has been developed that provides viewing S.M.A.R.T. HDD attributes, searches for bad sectors and tries to fix them.

Conclusions. Software that automate the process of monitoring the state of the hard disk has been developed.

The field of use are exploitation service of computer systems.

ЗМІСТ

	С.
Перелік скорочень та умовних познач 8	8
Вступ 9	9
1 Аналіз предметної області 11	11
1.1 Класифікація пошкоджених секторів та причини їх появи 11	11
1.2 Огляд існуючих програмних засобів, що вирішують аналогічні завдання 14	14
1.3 Розробка технічного завдання.....18	18
1.3.1 Підстава для розробки 18	18
1.3.2 Мета розробки та її призначення 18	18
1.3.3 Основні вимоги до програмного продукту 18	18
1.3.4 Вимоги до видів забезпечення і компонентів 20	20
1.3.5 Обмеження на установку і виконання програми 21	21
1.3.6 Вимоги до програмної документації 21	21
1.3.7 Умови експлуатації 21	21
1.3.8 Вимоги до маркування та упаковки 21	21
2 Матеріали та методи.....22	22
2.1 Огляд застосованих технологій та платформ 22	22
2.1.1 Технологія S.M.A.R.T. 22	22
2.1.2 Платформа Microsoft .NET Framework. 25	25
2.2 Огляд можливостей Windows Forms 28	28
2.3 Огляд можливостей операційної системи Windows 10 та мови програмування C#.....31	31
3 Основні рішення щодо реалізації компонентів системи 34	34
3.1 Розробка діаграми класів програми.....34	34
3.2 Розробка програмних модулів 35	35
3.3 Розробка інтерфейсу користувача 37	37
4 Керівництво програміста 41	41

4.1 Призначення і умови застосування програми	41
4.2 Основні вимоги до системи	41
4.3 Характеристики програми	42
4.4 Початкові та вихідні дані.....	43
4.5 Повідомлення програмісту	43
5 Керівництво оператора.....	44
5.1 Призначення програми	44
5.2 Умови використання програми.....	44
5.3 Виконання програми	44
5.4 Повідомлення оператору	52
Висновки.....	54
Перелік джерел посилання	56
Додаток А Текст програми	58
Додаток Б Копії слайдів презентації.....	82

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

DST	– Disk Self Test;
S.M.A.R.T.	– Self-Monitoring, Analysis and Reporting Technology;
WPF	– Windows Presentation Foundation;
ГГц	– гігагерц;
Гб	– гігабайт;
ЖД	– жорсткий диск;
Кб	– кілобайт;
МГц	– мегагерц;
ОС	– операційна система;
ПЗП	– постійний запам'ятовувальний пристрій;
ПК	– персональний комп'ютер;
ТЗ	– технічне завдання.

ВСТУП

Комп'ютерні технології тісно увійшли в життя людей. Досить складно уявити, як можна прожити в сучасному світі без ПК чи смартфона. Тепер кожен ранок середньостатистичної людини починається зі стандартного перегляду стрічки новин та перевірки особистої пошти. ПК повільно, але вірно, став постійним помічником в повсякденних справах.

Якщо є бажання підтримувати ПК у робочому стані – необхідно слідкувати за цілісністю його компонентів. З усіх компонентів ПК лише деякі вимагають більшого догляду і уваги, ніж жорсткий диск. Адже будь-які збої в роботі ЖД можуть призвести до втрати важливих даних з цього носія інформації. Тому моніторинг стану жорсткого диска є актуальною задачею.

Жорсткий диск на настільному комп'ютері або ноутбучі – це дуже активний компонент обчислювальної системи. Практично кожна дія користувача викликає активність жорсткого диска, включаючи фонові системні процеси, які користувач навіть не помітить. Надмірна активність жорсткого диска може викликати безліч проблем, потенційно сповільнюючи роботу системи і розряджаючи батарею ноутбука. Ось чому так важливо використовувати правильне програмне забезпечення для моніторингу жорсткого диска.

Для моніторингу стану ЖД у першу чергу необхідно використовувати S.M.A.R.T. технологію (самоконтроль, аналіз та звітування (SMART)) та періодично проводити перевірку диска на наявність пошкоджених («битих») секторів [1].

Ця система вбудована в більшість сучасних жорстких дисків і твердотільних накопичувачів і призначена для повідомлення про збої в роботі жорсткого диска або виникненні проблем. Виробники накопичувачів можуть використовувати свої власні підходи до SMART, але зазвичай вони вимірюють аналогічні показники продуктивності, такі як частота помилок читання, механічний удар, температура жорсткого диска, час пошуку і т. д.

Операційна система Windows надає безкоштовний інструмент перевірки диска, здатний допомогти користувачеві в досягненні мети постійного моніторингу жорсткого диска. Однак, функціональні можливості цього інструменту обмежені [2].

Тому, метою даної дипломної кваліфікаційної роботи бакалавра є розробка програми, що дозволяє отримати більш докладну інформацію про стан і використання жорсткого диска, здійснює пошук пошкоджених секторів та намагається їх виправити.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Класифікація пошкоджених секторів та причини їх появи

Жорсткий диск – важливий, але далеко не ідеальний компонент персонального комп'ютера. У деяких користувачів він працює ненадійно, «розсипається», дані з нього губляться або істотно пошкоджуються. Однією з найбільш частих проблем з ЖД є поява на ньому так званих «поганих» (поганих) секторів, які стають «головним болем» для багатьох користувачів.

Пошкоджений сектор на жорсткому диску – це крихітна ділянка дискового простору, яка не відповідає на запити читання або запису.

Пошкоджені сектора можуть виникати як на традиційних магнітних жорстких дисках, так і на сучасних твердотільних (SSD) накопичувачах. Існує два типи пошкоджених секторів: перший виникає в результаті фізичного пошкодження, яке неможливо виправити, а інший – в результаті програмних помилок, які піддаються виправленню.

Таким чином, пошкоджені сектора згідно [3] можна розділити на «фізичні» і «логічні» збійні сектора або «жорсткі» і «м'які» збійні сектора.

Фізичний або жорстко пошкоджений сектор – це частка сховища на жорсткому диску, яка фізично пошкоджений. Головка жорсткого диска могла торкнутися цієї частини жорсткого диска і пошкодити її, частинка пилу могла осісти на цей сектор і зіпсувати його, чарунка флеш-пам'яті твердотільного накопичувача могла бути зношена, або на жорсткому диску могли бути інші дефекти або проблеми зносу, які привели до фізичного пошкодження сектора. Цей тип сектора не підлягає ремонту.

Логічний – або програмно-зіпсований сектор, це частка сховища на жорсткому диску, яка, по всій видимості, не працює належним чином. Операційна система могла спробувати прочитати дані на жорсткому диску з цього сектора і виявила, що код виправлення помилок (ECC) не відповідає вмісту сектора, що говорить про те, що щось не так. Такі частки можуть бути позначені як пошкоджені сектора, але їх можна виправити, перезаписавши диск нулями

або, як раніше, виконавши низькорівневе форматування. Інструмент Windows Disk Check також може виправити такі пошкоджені сектори.

Розглянемо причини виникнення жорстко пошкоджених секторів.

Однією з причин може бути те, що можливо жорсткий диск вже був доставлений з заводу з пошкодженими секторами. Сучасні технології виробництва не є досконалими, і в усьому є похибка. Ось чому твердотільні накопичувачі часто поставляються з дефектними блоками.

У сучасних жорстких дисках розташовано кілька запасних секторів. Мікропрограма контролера диска визначає пошкоджені сектори і перепризначає їх на інший фізичний сектор. При виявленні збійного сектора відбувається автоматичне перепризначення; цей автоматичний процес зазвичай відбувається при перезапису сектора. Сектор ніколи не використовується повторно. Замість цього контролер дисководу просто видаляє адресу сектора зі списку придатних для використання місць зберігання в ПЗП. Такий підхід був відомий раніше як "гарячий ремонт" – ремонт на ходу.

На твердотільному накопичувачі природний знос в кінцевому підсумку призведе до того, що коли закінчується додаткова пам'ять твердотільного накопичувача, ємність накопичувача починає зменшуватися, оскільки сектори стають нечитабельними.

На традиційному магнітному жорсткому диску пошкоджені сектори можуть бути викликані фізичним пошкодженням. Жорсткий диск міг мати виробничий дефект, або природний знос міг привести до зносу частини жорсткого диска. Також диск може впасти, в результаті чого головка жорсткого диска торкнулася диска і пошкодила деякі з секторів, можливо проникнення повітря в закриту область жорсткого диска і частинки пилу могли пошкодити диск. Існують і інші можливі причини [4]:

- перегрів. Жорсткі диски надійно працюють при температурі 20 ~ 25 °С. Таким чином, через перегрів стабільність кварцового генератора часу часто змінюється. Ці зміни також можуть викликати пошкодження внутрішнього обладнання. Внаслідок цього, необхідно розміщувати жорсткий диск далеко від

місць нагріву і вживати ефективних заходів по його охолодженню;

- неправильне виключення комп'ютера. Жорсткий диск під час роботи знаходиться в режимі високошвидкісної роботи. В цьому випадку головка приводу може перебувати в різних місцях на диску для читання даних. Таким чином, якщо користувач раптово вимкне комп'ютер, головка приводу буде змушена повернутися на місце парковки. При цьому, головка диска може зачепити диск, що може безпосередньо привести до пошкодження секторів;

- статистична електрика. Коли користувачеві потрібно доторкнутися до диску, він має брати його за обидві сторони і не торкатися друкованої плати. Це необхідно внаслідок того, що якщо при дотику до диска виникає статична електрика, вона може пошкодити багато комп'ютерних компонентів, включаючи жорсткі диски.

Логічні «биті» сектори з'являються в результаті програмних помилок. Наприклад, якщо вимикається електроживлення в момент запису на ЖД комп'ютера, запис даних в сектор переривається на середині операції. У більшості випадків це призводить до появи секторів з даними, які не проходять перевірку запису даних. Такі сектори відзначаються як збійні. Віруси чи інше шкідливе програмне забезпечення також можуть послужити причиною системних помилок і появи битих секторів.

Симптоми появи битих секторів наступні:

- система завантажується повільно;
- система гальмує (зависає) при читанні і запису даних з ЖД;
- система взагалі відмовляється завантажуватися (досить часто – в середині процесу);
- іноді комп'ютер перезавантажується без причини;
- в процесі роботи ОС регулярно виникають різні помилки.

Одним з найкращих способів перевірки жорсткого диска на наявність битих секторів є використання функціональних можливостей спеціалізованих системних утиліт.

1.2 Огляд існуючих програмних засобів, що вирішують аналогічні завдання

Деякі виробники ЖД розробляють спеціалізовані утиліти перевірки тільки для жорстких дисків власного виробництва. Прикладом є діагностична утиліта Data LifeGuard для ОС Windows фірми Western Digital. Вона здійснює перевірку як внутрішніх, так і зовнішніх дисків даного виробника. Крім того, вона може надати користувачеві інформацію про модель і серійні номери накопичувачів, підключених до системи, а також наступну інформацію після виконання різних тестів [5]:

- Quick Test. Він виконує швидку самоперевірку SMART накопичувача для збору і перевірки інформації Data Lifeguard, що міститься на накопичувачі;
- Extended Test. Виконує повне сканування носія для виявлення пошкоджених секторів. Перевірка може зайняти кілька годин в залежності від ємності диска;
- Write Zeros. Тест записує нулі на диск з опціями «Повне стирання» і «Швидке стирання». Файлова система і дані будуть втрачені;
- View Test Results. Відображає результати останнього тестування.

Одним з більш універсальних аналогів програми, що розроблюється є програма «CrystalDiskInfo» [6].

CrystalDiskInfo – безкоштовна утиліта з відкритим вихідним кодом, що розроблена програмістом з Японії Noriyuki Miyazaki. Призначена для діагностики роботи жорстких дисків ПК. У процесі роботи програми відображається загальна інформація про ЖД, ведеться моніторинг значень S.M.A.R.T., а також здійснюється постійний контроль температури диска.

Актуальна версія – 8.3.1 (11 жовтня 2019). Програма виконується під керуванням ОС Windows XP/Vista/7/8/8.1/10.

Недоліками є відсутність можливості провести перевірку ЖД на наявність «битих» секторів та відсутність україномовного інтерфейсу.

Інтерфейс програми наведено на рис. 1.1.

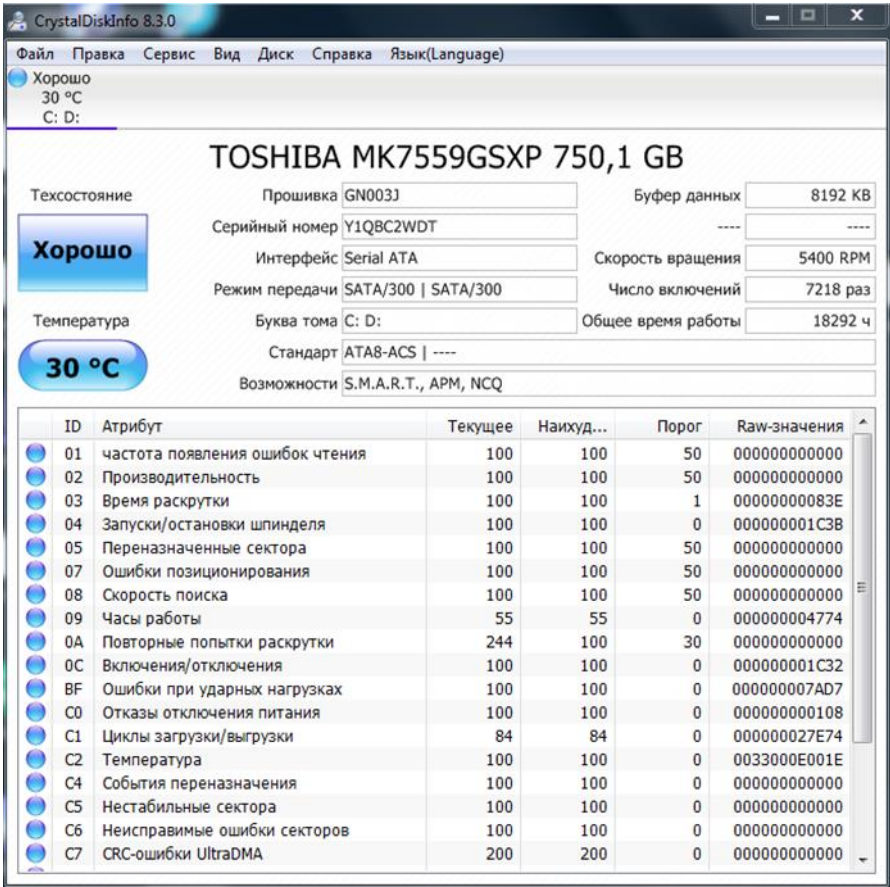


Рисунок 1.1 – Интерфейс програми «CrystalDiskInfo»

Наступним аналогом є програма «HD Tune Pro» [7].

HD Tune Pro – утиліта для моніторингу стану жорсткого диску і SSD, що включає в себе такі функції як визначення експлуатаційних якостей диска, сканування на помилки, перевірка стану здоров'я S.M.A.R.T., надійне видалення всіх даних і багато іншого. Розробник програми – EFD Software.

Перша версія HD Tune Pro з'явилася 12 лютого 2008 року. У версії 4.00, що вийшла 28 грудня 2009 року, з'явилася підтримка дисків більш 2 Тбайт і додані тест швидкого читання / запису і тест кеша. Остання версія 5.75 вийшла 27 серпня 2019 року, в яку додали підтримку контролера JMicron NVME для USB моста. Програма виконується під керуванням ОС Windows 7 та вище, має інтуїтивно зрозумілий інтерфейс, наявна функція перевірки рівня шуму ЖД.

Недоліками є відсутність української мови та необхідність купівлі ліцензії для користування цією програмою.

Інтерфейс програми «HD Tune Pro» наведено на рис. 1.2.

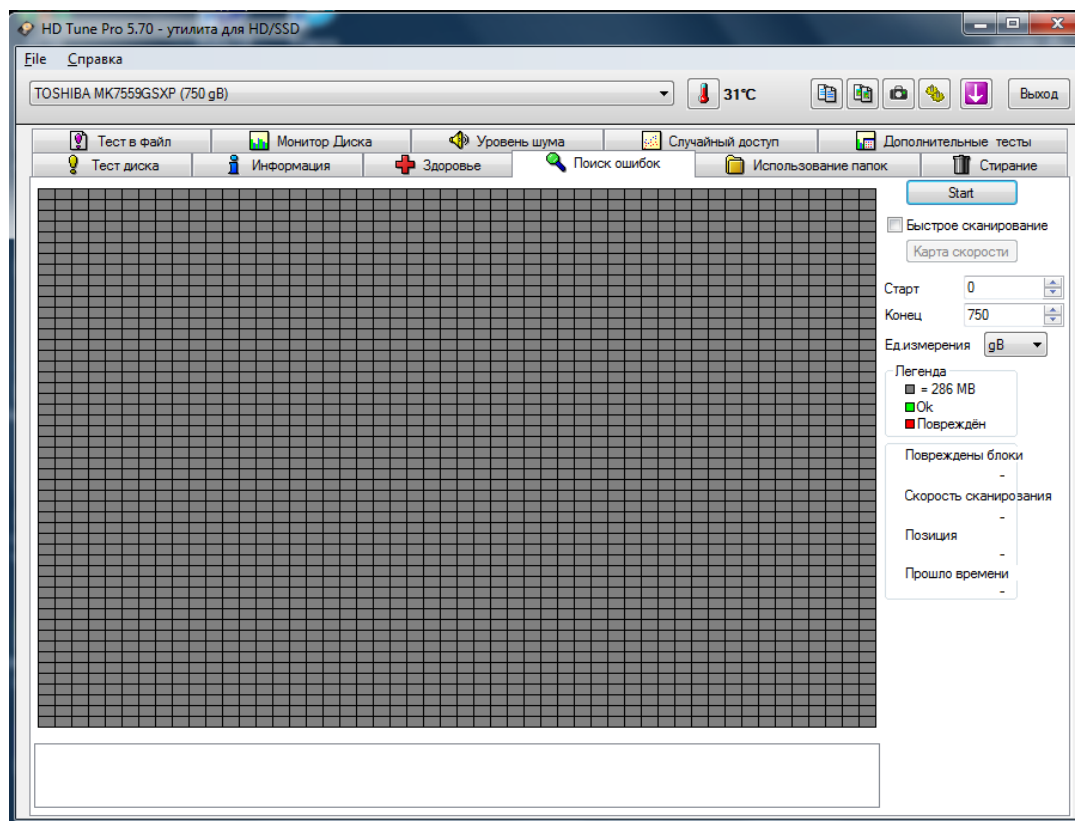


Рисунок 1.2 – Інтерфейс програми «HD Tune Pro»

Одною з найкращих безкоштовних програм для діагностики, дослідження, тестування і дрібного ремонту жорстких дисків, SSD-накопичувачів, карт пам'яті, а також будь-яких інших накопичувачів в операційній системі Windows є Victoria HDD/SSD [18].

Актуальна версія – Victoria 5.36 від 5 березня 2021 р. Застосунок має інтерфейси на російській, англійській та українській мовами (рис. 1.3). Підтримується технологія S.M.A.R.T. HDD/SSD російською та українською мовами, а також накопичувачі з інтерфейсом M.2/U.2 NVMe (для SSD накопичувачів).

У програму вбудований потужний сканер поверхні ЖД, який дозволяє продіагностувати накопичувач на наявність збійних ділянок, плаваючих

дефектів і помилок інтерфейсу. Victoria здатна протестувати більшість дисків на граничній швидкості, незалежно від їх типу. Особливість сканера – застосування спеціальних алгоритмів автоматичного налаштування таймаутів і розмірів блоку, що дозволяє ефективно перевіряти як повільні, так і швидкі диски.

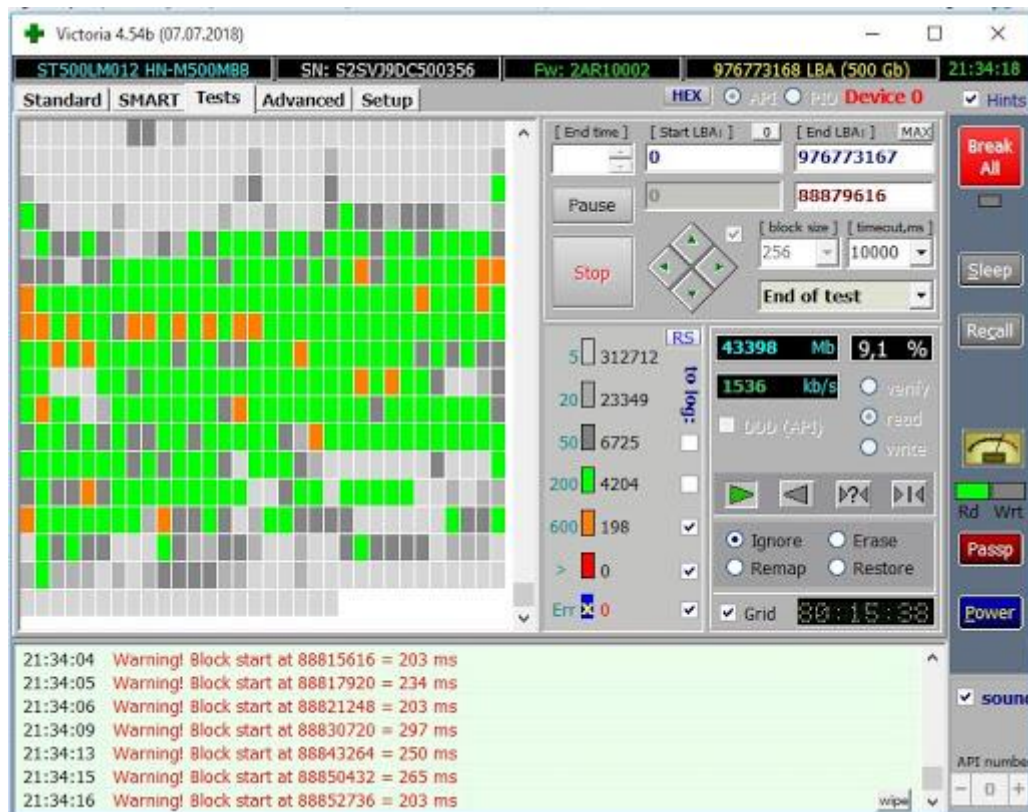


Рисунок 1.3 – Приклад результатів тестування жорсткого диска програмою Victoria HDD/SSD

Огляд аналогів показав, що більшість популярних застосунків для отримання інформації про жорсткий диск мають наступну функціональність:

- відображення інформації про загальний час роботи HDD;
- список S.M.A.R.T атрибутів та їх значення;
- можливість проведення перевірки HDD на наявність "битих" секторів.

Серед переважаючих недоліків спостерігаються перевантаженість інтерфейсу, відсутність підтримки української мови у більшості програм, а також те, що деяке програмне забезпечення є платним.

Проведений огляд існуючих програмних засобів, що вирішують аналогічні завдання послужив базою для розробки технічного завдання для програми моніторингу стану жорсткого диску.

1.3 Розробка технічного завдання

1.3.1 Підстава для розробки

Підставою для розробки служить завдання на дипломну кваліфікаційну роботу бакалавра на тему "Розробка програми для моніторингу стану жорсткого диску", затверджене наказом № 103 від 30 березня 2021 р. по НУ «Запорізька політехніка».

1.3.2 Мета розробки та її призначення

Метою створення програмного забезпечення є розробка програми, яка надавала б можливість отримувати інформацію про стан ЖД.

Програмне забезпечення, що розроблюється призначене для отримання інформації про стан ЖД та виконання пошуку пошкоджених секторів на диску.

1.3.3 Основні вимоги до програмного продукту

Сформулюємо вимоги до програми «HDD Easy Scanner», що розробляється.

Програмне забезпечення, що розробляється повинне:

- функціонувати під керуванням ОС Windows, версії 10;
- надавати можливість переглядати інформацію про загальний час роботи ЖД;
- надавати можливість моніторингу поточної температури ЖД;
- надавати можливість проведення перевірки ЖД на наявність пошкоджених секторів з відображенням результатів перевірки у вигляді карти,

що відображає стан поверхні ЖД. Карта повинна складатися зі 2200 (55x40) візуальних блоків, кожен з яких показує інформацію про певну перевірену ділянку ЖД;

- можливість переглядати список, зазначених у табл. 1.1, S.M.A.R.T. атрибутів та їх значення (current, worst, value);
- мати нескладний та дружній інтерфейс.

Таблиця 1.1 – Перелік необхідних S.M.A.R.T. атрибутів

Номер	Назва атрибуту
1	2
1	Помилки зчитування даних з диска
3	Час розкручування пакету дисків (мс)
4	Кількість запуску/зупинки шпинделя
7	Помилки позиціонування блоку магнітних головок
9	Час роботи (год)
10	Повторні спроби запуску ЖД
12	Кількість включень ЖД
13	Кількість помилок при зчитуванні
188	Кількість перерваних операцій через HDD тайм-аут
189	Кількість випадків запису при висоті головки вище стандартної
191	Кількість помилок через ударні навантаження
192	Відмова відключення живлення
193	Кількість циклів завантаження/розвантаження блоку головок
194	Температура для механічної частини диска
198	Невиправні помилки секторів
200	Кількість помилок під час запису
207	Сила струму при розкручуванні диску
220	Дистанція зміщення блоку дисків щодо шпинделя (струс диску)
223	Повтори вивантажень/завантажень блоку магнітних головок

Продовження табл. 1.1

1	2
226	Час завантаження головок (мс)
240	Час знаходження блока головок у робочому стані (год)
254	Захист від падіння

Вхідними даними, необхідними для функціонування програмного продукту, є поточна інформація про стан ЖД, яка буде автоматично отримуватися програмою.

Вихідними даними є наглядне відображення інформації про стан ЖД.

1.3.4 Вимоги до видів забезпечення і компонентів

Програма повинна працювати під керуванням ОС Windows, версії 10 з встановленим .NET Framework версії не нижче 4.6.1.

У якості інструментальної мови програмування для розробки даної програми слід використовувати мову програмування C# та інтегроване середовище розробки Visual Studio 2017.

Вимоги до технічного забезпечення для нормального функціонування програми обумовлені вимогами до ОС Windows 10:

- ПК з мікропроцесором Intel Pentium 1,0 ГГц та вище;
- наявність вільного місця на жорсткому диску 700 Кб (для програми) та 16 Гб для 32-розрядної ОС або 20 Гб для 64-розрядної ОС;
- оперативна пам'ять 1 Гб (для 32-розрядної версії) або 2 Гб (для 64-розрядної версії) ОС;
- відеоадаптер з DirectX 9 або новішої версії з драйвером WDDM 1.0;
- дисплей мінімум 800x600 пікселів;
- клавіатура і маніпулятор типу "миша".

1.3.5 Обмеження на установку і виконання програми

Програму, що розробляється в даній роботі можна встановити і перевірити на будь-якому ПК з ОС Windows, версії 10 з встановленим .NET Framework версії не нижче 4.6.1. Для можливості запуску програми необхідно мати права адміністратора.

Запуск програми «HDD Easy Scanner» має здійснюватися шляхом запуску файлу «HDD Easy Scanner.exe» з правами адміністратора.

1.3.6 Вимоги до програмної документації

Програмна документація повинна включати: керівництво програміста, керівництво оператора, пояснювальну записку на розроблений застосунок та вихідний код програми

1.3.7 Умови експлуатації

Програма повинна функціонувати під управлінням ОС Windows 10. Необхідне встановлення .NET Framework версії не нижче 4.6.1.

1.3.8 Вимоги до маркування та упаковки

Програма в своєму повному обсязі може бути записана на компакт-диск ємністю 900 Кб. На упаковці має бути назва програми «HDD Easy Scanner».

2 МАТЕРІАЛИ ТА МЕТОДИ

2.1 Огляд застосованих технологій та платформ

2.1.1 Технологія S.M.A.R.T.

Технологія самоконтролю, аналізу та звітності (SMART) - це технологія виявлення, моніторингу та обслуговування несправностей, яка використовується комп'ютерами для надання розширених повідомлень про збої ЖД.

У 1992 році фірма IBM запустила SMART для своїх серверів дискових масивів AS/400 для виявлення і прогнозування помилок жорсткого диска.

SMART можна розглядати як інтерфейс між базовою системою вводу/виводу (BIOS) і жорстким диском, який працює з помилками дисків, які виникають через вік і/або через механічний знос. SMART може виявляти несправності після оцінки «битих» секторів на фізичній поверхні диска, голівці диска та інших механічних деталях.

S.M.A.R.T вимірює поточний стан жорсткого диска в порівнянні з еталоном, встановленим виробником. Якщо який-небудь з тестів не буде виконано, жорсткий диск буде реєструвати його, і він буде підсумовуватися, а частота цих помилок може бути повідомлена як неминучий збій.

Існує два типи відмов, з якими можуть зіткнутися жорсткий диск і твердотільний накопичувач: передбачувані збої і непередбачувані збої.

Передбачувані збої - це збої, які з'являються під час використання. Ці збої включають пошкодження диска, збій читання/запису, відмову двигуна у випадку жорстких дисків. У SSD ці збої можуть бути через зношування з плином часу або великої кількості невдалих спроб стирання. Проблема з часом посилюється, і диск виходить з ладу.

Грунтуючись на рівні зносу, SMART може прогнозувати термін служби диска, що залишився, що дозволяє користувачеві створювати резервні копії даних і/або замінювати диск. Однак SMART не може виявляти непередбачувані збої, такі як відмови електронних компонентів, раптовий стрибок напруги або фізичне втручання [8].

Жорсткі диски можуть виконувати різні типи тестів S.M.A.R.T для різних цілей:

- короткий літній час (Short summer time);
- довгий/розширений літній час (Long/extended summer time);
- транспортування DST (DST transport);
- вибіркового літнього часу (Selective summer time).

Кожен раз, коли комп'ютер запускається, виконується короткий тест самотестування ЖД (Short Disk Self (DST) Test). Даний тест – це складова S.M.A.R.T технології, яка запускається при запуску ПК і виконує коротку перевірку основних компонентів жорсткого диска.

Тест DST фокусує свою увагу виключно на основних компонентах, таких як головки читання/запису, ПЗП, плата управління, пластини і двигун. Самотестування гарантує, що ці основні компоненти працюють належним чином, і видасть попередження, якщо який-небудь з них не працює належним чином. Ця перевірка зазвичай займає всього кілька хвилин.

Тест «Короткий перехід на літній час» автоматично запускається утилітою, встановленою на жорсткий диск виробником. Він запускається відразу після запуску самотестування при включенні ПК під час завантаження. Якщо даний тест пройдено, комп'ютер завантажить операційну систему.

В ході довгого/розширеного самотестування диска основні компоненти будуть перевірені разом з даними, що зберігаються на пластині ЖД. Якщо Long DST виявляє пошкоджені області на диску, він спробує перерозподілити і перепризначити пошкоджені сектори. Це гарантує, що жорсткий диск більше не розподілить інформацію в ці погані області. Жорсткий диск не можна використовувати до завершення даного тесту.

Тест транспортування DST – це тест, призначений для перевірки ушкоджень, які могли виникнути під час транспортування жорсткого диска від виробника комп'ютера до користувача. Він доступний тільки на дисках ATA і зазвичай займає кілька хвилин.

Що стосується тесту «Вибірковий літній час», то деякі диски дозволяють тестувати тільки частину поверхні пластин диску.

Більш детальну інформацію про стан накопичувача можна отримати шляхом аналізу статистики набору S.M.A.R.T. атрибутів. Тобто, накопичувач має набір атрибутів із визначеними пороговими значеннями. Атрибут вважається правильним, якщо його значення більше або дорівнює граничному значенню. Якщо це не так для критичного атрибута, прогнозується відмова, пристрій вважається несправним і його слід негайно замінити (атрибут визначає проблему). Якщо поріг дорівнює 0 для будь-якого атрибута, цей атрибут не може передбачити збій (оскільки значення не може бути менше 0). Кожен виробник накопичувача сам визначає цей набір атрибутів і встановлює граничні значення, за які атрибути не повинні виходити при нормальній роботі.

Зазвичай атрибути містять наступні поля [9]:

- ідентифікатор (значення атрибута);
- дані (необроблені). В цьому полі зберігаються необроблені значення, що отримані від датчика або лічильника. Потім ці дані обробляються за алгоритмом, розробленим виробником ЖД;
- поріг, тобто граничне значення для атрибута (відмова);
- значення, тобто поточне відносне «здоров'я» атрибута. Це число розраховується алгоритмом з використанням необроблених даних;
- найгірше (найменше) значення, коли-небудь виявлене за попередній термін служби ЖД.

S.M.A.R.T. атрибути можна розділити на дві групи:

- параметри, що відображають процес природного старіння твердого диска (число обертів шпинделя, число переміщень головок, кількість циклів ввімкнення-вимкнення);
- поточні параметри накопичувача (висота головок над поверхнею диска, число секторів, що перепризначувалися, час пошуку доріжки і кількість помилок пошуку).

Дані зберігаються в шістнадцятковому вигляді, у так званому «Raw value» форматі, а потім перераховуються в «Value», значення, що символізує надійність щодо деякого еталонного значення. Зазвичай «Value» має значення в діапазоні від 0 до 100 (деякі атрибути мають значення від 0 до 200 і від 0 до 253).

Перелік S.M.A.R.T. атрибутів, значення яких повинна визначати програма, що розроблюється наведено у табл. 1.1.

2.1.2 Платформа Microsoft .NET Framework

Microsoft .NET – це середовище розробки програмного забезпечення, яке розроблено і підтримується компанією Microsoft для спрощення розробки як десктопних так і веб застосунків. Це популярна безкоштовна платформа, яка в даний час використовується для множини застосунків різних типів, оскільки вона надає середу програмування для більшості етапів розробки програмного забезпечення. .NET найкраще підходить компаніям, які шукають широкий спектр функцій, таких як вебслужби, програмне забезпечення для настільних комп'ютерів та підтримка хмарної інфраструктури.

Платформу .NET Framework було розроблено для досягнення наступних цілей [10]:

- забезпечення узгодженого об'єктно-орієнтованого середовища програмування, незалежно від того, чи зберігається і виконується об'єктний код локально, поширюється через Інтернет, або виконується віддалено;
- забезпечення середовища виконання коду, яке зводить до мінімуму розгортання програмного забезпечення і конфлікти версій, сприяє безпечному виконанню коду, включаючи код, створений невідомою або частково довіреною третьою стороною, усуває проблеми з продуктивністю скриптових або середовищ, що інтерпретуються;
- забезпечення однаковості взаємодії з розробниками для самих різних типів застосунків, таких як застосунки для Windows і вебзастосунки;

– побудова на основі галузевих стандартів, щоб код, заснований на .NET Framework, міг інтегруватися з будь-яким іншим кодом.

Платформа .NET Framework складається з загальномовного середовища виконання (CLR) і бібліотеки класів .NET Framework. Серед CLR є основою .NET Framework. Середовище виконання виступає в якості агента, який управляє кодом під час виконання, надаючи основні послуги, такі як управління пам'яттю, управління потоками і віддалену взаємодію, а також забезпечуючи строгу типізацію і інші форми точності коду, які сприяють безпеці та надійності. Фактично, концепція управління кодом є фундаментальним принципом середовища виконання. Код, орієнтований на середовище виконання, відомий як керований код, а код, що не орієнтований на середовище виконання, відомий як некерований код.

Бібліотека класів являє собою комплексну об'єктно-орієнтовану колекцію багаторазових типів, які програміст використовує для розробки застосунків, починаючи від традиційних застосунків командного рядка або графічного інтерфейсу користувача (GUI) до застосунків, заснованих на останніх інноваціях, що надаються ASP.NET, таких як Web Forms і вебслужби XML.

Незважаючи на те, що .NET вважається однією з найсильніших інженерних платформ завдяки розгалуженій інфраструктурі та перевірній історії розробки продуктів, ця платформа має ряд проблем [11]:

– обмежена об'єктно-реляційна підтримка. Як було зазначено вище, .NET використовує модель об'єктно-орієнтованого програмування. Ця модель зосереджена на об'єктах, а не на «діях» та даних. Підтримка розробки програмних застосунків, орієнтованих на дані, у .NET Framework та Core забезпечується за допомогою Entity Framework. Entity – це об'єктно-реляційний перетворювач (ORM), який пов'язує об'єктно-орієнтовані .NET Framework і реляційні (SQL) бази даних. Деякі інженери вважають, що Entity Framework недостатньо гнучка і може не підтримувати всі доступні конструкції баз даних. Це також означає, що є вірогідність, що в якийсь момент Entity Framework може

не підтримувати нові проєкти баз даних. Інша проблема полягає в тому, що в кінцевому підсумку від фреймворка може відмовитися Microsoft;

- вартість ліцензування. Створення застосунків .NET не з дешевих, незалежно від технологій з відкритим вихідним кодом. В основному витрати полягають у покупці Visual Studio IDE та інших додаткових послуг для спільної роботи і забезпечення якості, які Microsoft пропонує для спрощення роботи програміста. На сьогоднішній день базова версія Visual Studio буде коштувати вам 1999 доларів в рік за одного нового інженера і 799 доларів за продовження підписки. Корпоративний випуск, який включає в себе різні хмарні середовища, корпоративну мобільність, функції контролю якості та багато іншого, буде коштувати близько 6000 доларів в перший рік і 2569 доларів за продовження. Для некомерційних та освітніх цілей також існує безкоштовне співтовариство Visual Studio;

- нестабільність нових релізів. Ця проблема властива не тільки .NET, але і іншим продуктам Microsoft. У багатьох недавніх продуктах відсутня належна документація, підтримка, стабільність і вони зазнали кардинальних змін у подальшому. Наприклад, у одному з проєктів AltexSoft програмісти були змушені розробити власну архітектуру даних, оскільки продукт мав ряд технічних проблем, які могли поставити під загрозу весь проєкт;

- витік пам'яті. Проблема витоку пам'яті у тому, що фреймворк не звільняє пам'ять, яка йому більше не потрібна. Хоча в .NET є збирач сміття для такого роду проблем, інженерам все ж доводиться докладати додаткових зусиль для правильного управління ресурсами. І витоки можуть продовжують рости в міру масштабування застосунку. Слід зазначити, що програміст зможе знайти докладні і дієві рекомендації з виявлення витоку в .NET;

- утруднений перехід на .NET Core. .NET Core – це наступний великий крок у розвитку платформи Microsoft. Як завжди буває, перехідний період може бути довгим і важким. Кількість застосунків, створених з використанням старої .NET Framework, перевищує кількість застосунків з .NET Core. І це не обов'язково означає, що всі вони зазнають цей перехід. У корпоративному

середовищі перехід може бути не таким пріоритетним. Проте, ці системи потребують підтримки. Інша пов'язана з цим проблема полягає в тому, що у .NET Framework було багато підтримуючих проєктів. Наприклад, багато розробників висловили заклопотаність з приводу відмови від Windows Communication Foundation (WCF), середовища виконання та набору API для сервіс-орієнтованих програм, що використовуються багатьма практиками .NET. Microsoft вирішила не переносити WCF на Core. Замість цього вона залучила до цього зусилля спільноти, оскільки один з членів команди .NET Foundation пожертвував альтернативу з відкритим вихідним кодом, Core WCF. Але Microsoft не керує цим безпосередньо і не забезпечує повну сумісність з вихідним WCF. Розробникам доведеться витратити додаткові зусилля, щоб завершити перехід.

Платформу Microsoft .NET було обрано через її зазначені вище переваги і високу інтеграцію до Visual Studio.

2.2 Огляд можливостей Windows Forms

Для забезпечення нескладного інтерфейсу, відповідно до п. 1.4.3 технічного завдання, а також підвищення якості і прискорення розробки програми застосовано компонент Windows Forms. Для застосування компонента «Windows Forms» було використано програмну платформу Microsoft .NET (див. п. 2.1.2), що включає цей компонент в базовому наборі.

Windows Forms – інтерфейс програмування застосунків, що відповідає за побудову графічного інтерфейсу користувача і є частиною Microsoft .NET Framework.

Windows Forms надав розробникам .NET знайомий спосіб написання призначених для користувача інтерфейсів, оскільки Windows Forms був рівнем вище ніж Win32 API. Програміст може написати як базовий код, так і призначений для користувача інтерфейс на одній мові .NET, найбільш популярними з яких є мова C# та VB.NET.

З появою .NET також було випущене середовище розробки Visual Studio. Visual Studio включає конструктор користувацького інтерфейсу «перетягування» (drag-and-drop) для Windows Forms. Це значно підвищило продуктивність розробників, навіть якщо вони не були експертами по проектуванню інтерфейсу користувача.

Windows Forms має тільки два основних випуску: .NET 1 і .NET 2.0. З тих пір час від часу виправлялися помилки або поліпшувалася висока роздільна здатність, але розробка всіх функцій фактично припинилася. Проте, ця технологія, яку часто вважають тонкою оболонкою для власне користувацького інтерфейсу Windows, ніколи офіційно не вважалася застарілою. Продовжували розроблятися нові проекти, особливо внутрішні бізнес-застосунки, де цінувалися простота і продуктивність Windows Forms.

Windows Forms отримав кілька поліпшень у наступних версіях .NET Framework. У 2006 році Microsoft представила .NET 3.0, який включав аналог Windows Forms у вигляді Windows Presentation Foundation (WPF). WPF використовував інший підхід.

WPF відділив поведінку елемента управління призначенням для інтерфейса користувача від його зовнішнього вигляду. У той час це було революційно. WPF також приніс у світ програмування XAML, тобто розширювану мову розмітки застосунків.

Комбінація мови .NET (наприклад, C # або VB.NET) для логіки і XAML для розмітки продовжилася в Windows 8 і Windows 10. Остання платформа користувацького інтерфейсу робочого столу від Microsoft, універсальна платформа Windows або UWP, заснована на XAML.

Але в .NET 5.0 почалася серйозна розробка нових функцій Windows Forms. Здебільшого це спонсорувалося спільнотою, а не компанією Microsoft. Заявлено про наступні нові можливості середовища виконання Windows Forms в .NET 5.0 [12]:

- новий елемент управління TaskDialog. Діалог завдань був введений в Windows Vista як заміна діалогового вікна повідомлення, яке Windows

використовувала з 90-х років. На відміну від вікна повідомлення, в якому дозволені тільки заздалегідь певні пари параметрів, такі як OK/Cancel або Yes/No, розробники можуть надати свій власний текст для параметрів в діалоговому вікні завдання. Інші функції включають окремі текстові блоки «основна інструкція», «зміст», «додаткова інформація» і «нижній колонтитул», радіокнопки, прапорець з міткою, що часто використовується для «Більше не питати», індикатор виконання з додатковим таймером і автоматичним закриттям, можливість мінімізувати вікно. Хоча діалогове вікно завдань є частиною операційної системи, воно не відображається в Windows Forms від Microsoft. Однак його можна було використовувати через виклики бібліотеки Win32 (наприклад, `p/invoke`);

- поліпшення `ListView`. Вони включають можливість згорнути/розгорнути групу `ListView`, нижні колонтитули групи `ListView`, субтитри групи `ListView`, групові завдання `ListView` та зображення заголовка групи `ListView`. Як і у випадку з діалоговим вікном завдань, ця функція потенційно може бути доступна через виклики Win32;

- поліпшення `FileDialog`. `FileDialog` отримав новий API: `FileDialog.ClientGuid`. Діалогове вікно файлу Windows дозволяє програмі, що викликає, пов'язати ідентифікатор GUID із постійним станом діалогового вікна. Стан діалогового вікна може включати такі фактори, як остання відвідана папка та положення та розмір діалогового вікна. Зазвичай цей стан зберігається на основі імені виконуваного файлу. Вказавши GUID, програма може мати різні збережені стани для різних версій діалогового вікна в рамках однієї програми;

- поліпшення продуктивності;
- поліпшення і виправлення спеціальних можливостей. Модернізація спеціальних можливостей Windows Forms також є пріоритетною задачею. Це включає в себе підтримку автоматизації користувацького інтерфейсу різних елементів управління, підтримку шаблонів елементів управління `LegacyIAccessible`, а також підтримку шаблонів елементів керування `Text` і `TextRange`;

- підтримка Visual Basic.

Інтерфейс програмування застосунків Windows Forms було обрано через його відносно легке налаштування і адаптованість до ОС Windows.

2.3 Огляд можливостей операційної системи Windows 10 та мови програмування C#

З більш ніж одним мільярдом установок (за даними корпорації Microsoft) ОС Windows 10 тепер займає більше 78 відсотків ринку Windows [13] (для десктопних застосунків) і випереджає Windows 7 більш ніж на 62 % (див рис. 2.1).

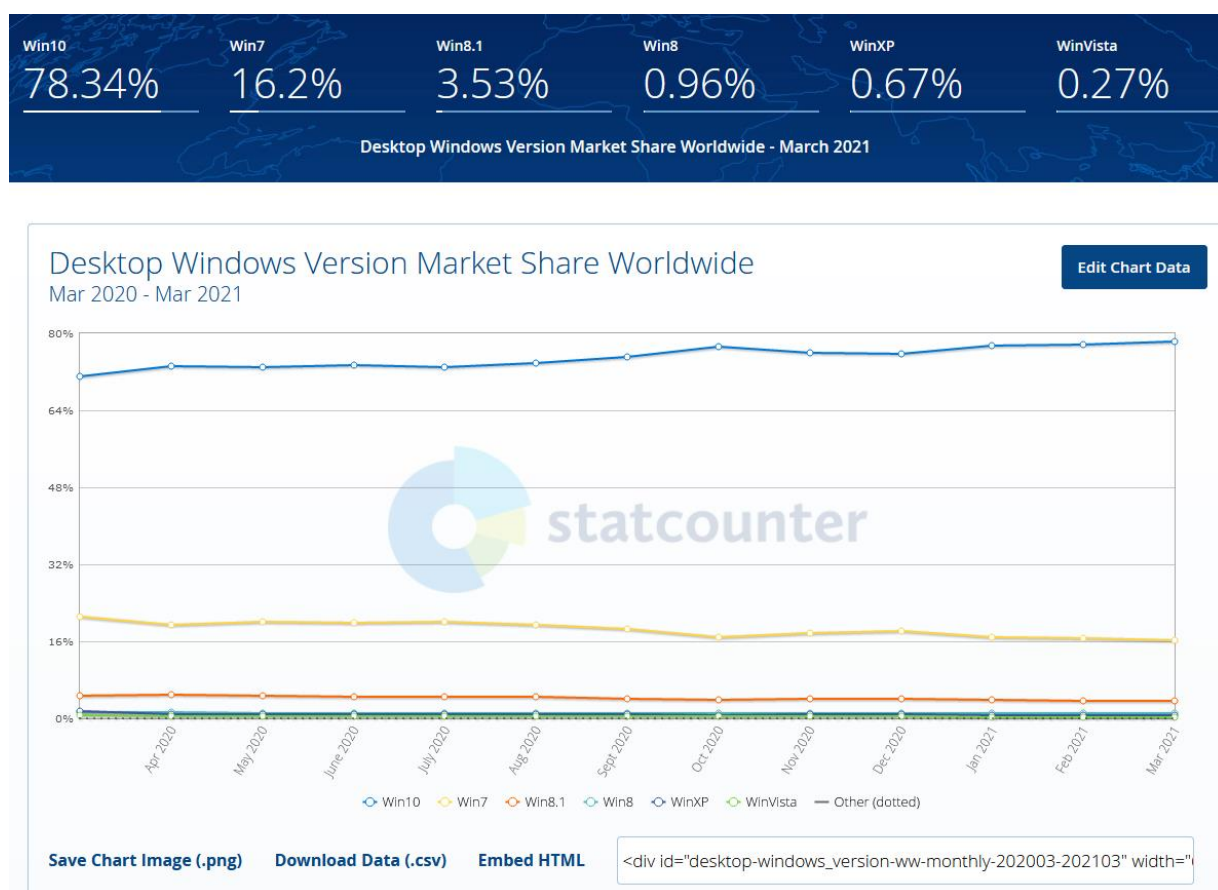


Рисунок 2.1 – Світова частка ринку настільних версій ОС Windows

Крім того, сімейство Windows в цілому займає більше 87 відсотків світового ринку операційних систем для настільних ПК [14]. Для порівняння, на

всі версії MacOS від Apple припадає менше 20 відсотків комп'ютерів в США і менше половини від цього показника в усьому світі. Linux в цілому отримує близько двох відсотків, а Ubuntu і Chrome OS у всьому світі займають менше одного відсотка.

Важливо, що багато оновлень безпеки, призначених для захисту платформи Windows від відомих вразливостей, реалізовані в передостанній версії Windows 10 – 20H2 (жовтень 2020 р.).

На відміну від поновлення за травень 2020 року, яке включає в себе абсолютно новий набір функцій і поліпшень продуктивності, це оновлення є відносно незначним. Його основна увага приділяється впровадженню виправлень в попереднє оновлення.

Однак це не означає, що оновлення 20H2 не містить власних нових функцій. Наведемо перелік найбільш суттєвих з них [15]:

- повна інтеграція в ОС браузера Microsoft Edge на основі Chromium. Edge поки не вміє використовувати розширення і не має інтегрованого блокувальника реклами. Однак цей браузер може показувати актуальні новини та рекомендації по іграм та застосункам, погоду і запропонувати пошук на головній сторінці під час запуску програми. Нова версія Edge на основі Chromium стає вбудованим веббраузером, який підтримує не тільки сам браузер Edge, але і застосунки магазину Windows, що вимагають доступу в Інтернет. Це означає, що обидва вони стають більш сумісними з вебсайтами, оскільки основний код рендеринга той же, що і в Google Chrome. Також варто відзначити, що Edge працює не тільки в Windows, але і на Android, iOS, macOS, а тепер навіть і в Linux;

- підтримка браузером Edge комбінації клавіш Alt-Tab. Комбінація клавіш тепер працює з вебсайтами так само, як вона завжди працювала з застосунками;

- нове меню Пуск;

– додаткова підтримка ігрових моніторів з високою частотою оновлення, а також поліпшена панель задач і додавання логотипів додатків в повідомлення Windows 10.

Враховуючи той факт, що ОС Windows 10 є найбільш поширеною ОС і одночасно, найновішою в лінійці ОС Windows, програма буде розроблятися саме під дану ОС.

Для створення настільних застосунків використовується багато фреймворків та мов. Їх можна розробити на категорії за платформами або кросплатформними технологіями, що ними підтримуються, а також за використанням програмних програм та бібліотек.

Оскільки як ОС, на якій буде працювати програма, було обрано Windows, було прийнято рішення для розробки застосунку використовувати найбільш адаптовану мову програмування для цієї ОС – мову C# [16], [17].

Ця мова загального призначення включає строгу типізацію, декларативне, функціональне об'єктне і компонентно-орієнтоване програмування. C# приблизно синтаксично схожий на Java, був розроблений Microsoft і використовується у всіх продуктах Microsoft. В основному мова програмування C# використовується для розробки настільних застосунків, а найостаннішим настільним застосунком є Windows 8 і 10.

Крім того, він є частиною .NET, тому використовує суміжні мови. Наприклад, ASP в застосунках і веброзробках. C# також використовується для широкого спектру корпоративних застосунків, що працюють на .NET.

3 ОСНОВНІ РІШЕННЯ ЩОДО РЕАЛІЗАЦІЇ КОМПОНЕНТІВ СИСТЕМИ

3.1 Розробка діаграми класів програми

Для реалізації функціональних вимог до програми, зазначених у п. 1.3 технічного завдання було розроблено шість класів, які взаємодіють між собою. На рис. 3.1 наведено діаграму класів програми «HDD Easy Scanner».

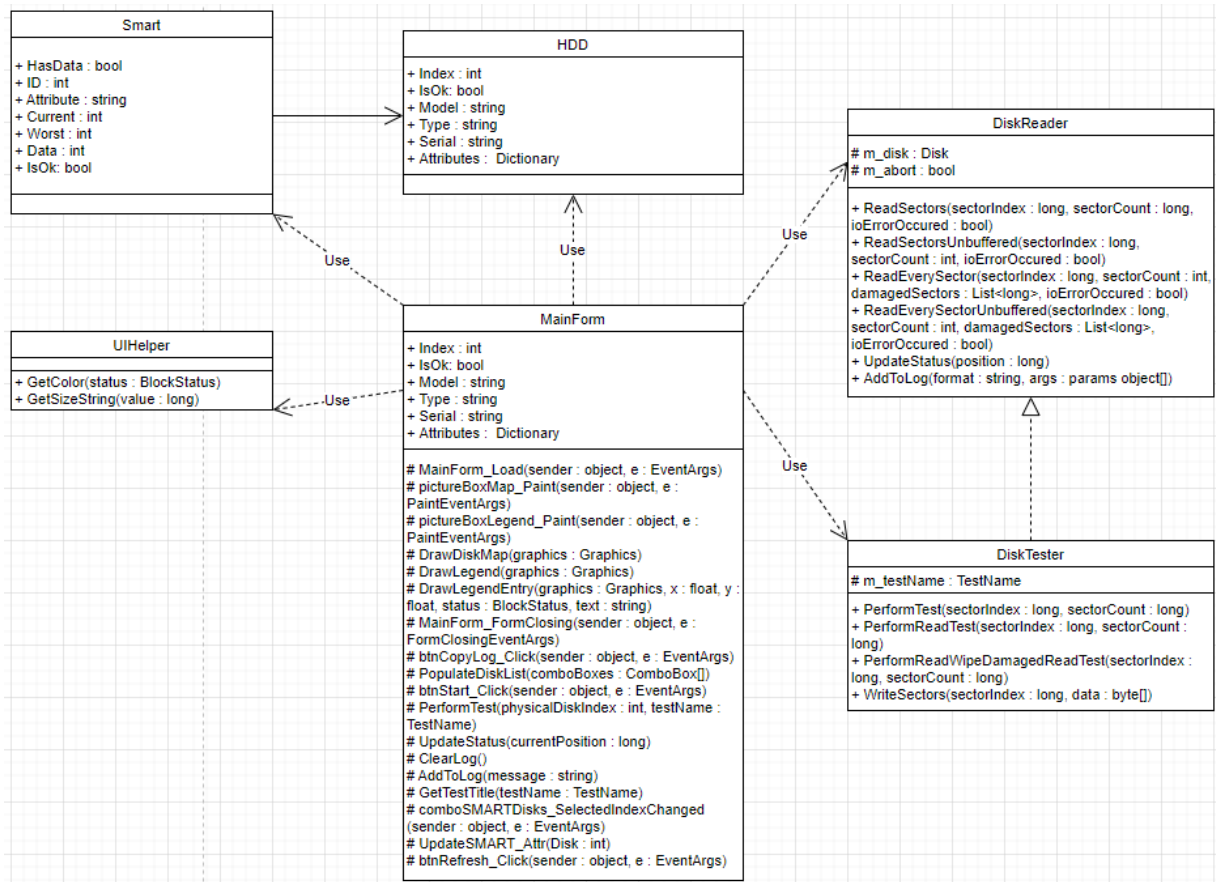


Рисунок 3.1 – Діаграма класів «HDD Easy Scanner»

Клас «MainForm» є основою програми. Він відповідальний за координацію всіх її частин.

Клас «Smart» відповідає за збереження даних про S.M.A.R.T. атрибут.

Клас «HDD» відповідає за збереження даних про ЖД.

Клас «DiskReader» відповідає за отримання основної інформації про ЖД.

Клас «DiskReader» відповідає за тестування ЖД на наявність «битих» секторів.

Більш докладне пояснення основних методів та функцій класів наведено в п. 3.2.

3.2 Розробка програмних модулів

Нижче будуть наведені основні класи, що реалізують програму застосунку, з прототипами основних методів. Повний код основних класів наведено у Додатку А.

Розглянемо код та опис (у вигляді коментарів) методів класу «MainForm»:

```
public class MainForm
{
    public MainForm() // головна функція, конструктор форми
    {
        // завантаження форми
    }

    private void MainForm_Load(object sender, EventArgs e)
    {
        // викликається в момент відображення карти диску
    }

    private void pictureBoxMap_Paint(object sender, PaintEventArgs e)
    {
        // викликається в момент відображення легенди карти
    }

    private void pictureBoxLegend_Paint(object sender, PaintEventArgs e)
    {
        // функція для відображення карти диску
    }

    private void DrawDiskMap(Graphics graphics)
    {
        // функція для відображення легенди карти
    }

    private void DrawLegend(Graphics graphics)
    {
        // функція для відображення елементів легенди карти
    }

    private void DrawLegendEntry(Graphics graphics, float x, float y,
        BlockStatus status, string text)
    {
        // функція викликається в момент закриття головної форми
    }

    private void MainForm_FormClosing(object sender, FormClosingEventArgs e)
    {
        // функція викликається при натисканні кнопки «Скопіювати лог»
    }

    private void btnCopyLog_Click(object sender, EventArgs e)
    {
        // функція для отримання переліку встановлених ЖД
    }

    private void PopulateDiskList(ComboBox[] comboBoxes)
    {
        // функція викликається при натисканні кнопки «Старт»
    }

    private void btnStart_Click(object sender, EventArgs e)
    {
        // функція для початку тесту
    }

    private void PerformTest(int physicalDiskIndex, TestName testName)
    {
        // функція для оновлення статусу комірки на карті ЖД
    }
}
```

```

private void UpdateStatus(long currentPosition)
// функція для очищення логу
private void ClearLog()
// функція для додавання тексту до логу
private void AddToLog(string message)
// функція для отримання назви тесту
private static string GetTestTitle(TestName testName)
// функція для отримання S.M.A.R.T. атрибутів
private void UpdateSMART_Attr()
// функція викликається при зміні пункту у списку дисків
private void comboSMARTDisks_SelectedIndexChanged(object sender,
EventArgs e)
// функція викликається при натисканні кнопки «Оновити»
private void btnRefresh_Click(object sender, EventArgs e)

```

Код та опис методів класу «UIHelper»:

```

public class UIHelper
// функція для отримання кольору відповідно до статусу комірки на карті ЖД
    public static Color GetColor(BlockStatus status)
// функція для отримання розміру байтів
    public static string GetSizeString(long value)

```

Код та опис методів класу «DiskReader»:

```

public class DiskReader
// функція для зчитування секторів
    ReadSectors(long sectorIndex, long sectorCount, out bool ioErrorOccured)
// допоміжна функція для зчитування секторів
    ReadSectorsUnbuffered(long sectorIndex, int sectorCount, out bool
ioErrorOccured)
// функція для зчитування певного сектору
    ReadEverySector(long sectorIndex, int sectorCount, out List<long>
damagedSectors, out bool ioErrorOccured)
// допоміжна функція для зчитування певного сектору
    ReadEverySectorUnbuffered(long sectorIndex, int sectorCount, out
List<long> damagedSectors, out bool ioErrorOccured)
// функція для оновлення статусу блоку на карті ЖД
    UpdateStatus(long position)
// функція для додавання інформації до логу
    AddToLog(string format, params object[] args)

```

Код та опис методів класу «DiskTester»:

```
public class DiskTester
// функція для обрання певного методу тесту
    PerformTest(long sectorIndex, long sectorCount)
// функція для методу пошуку "битих" секторів
    PerformReadTest(long sectorIndex, long sectorCount)
// функція для методу пошуку та виправлення "битих" секторів
    PerformReadWipeDamagedReadTest(long sectorIndex, long sectorCount)
// функція для запису секторів
    WriteSectors(long sectorIndex, byte[] data)
```

3.3 Розробка інтерфейсу користувача

При розробці інтерфейсу застосунку враховано одну з вимоги пункту 1.3.3 ТЗ – наявність нескладного та дружнього інтерфейсу. Це забезпечено єдиним стилем програми, близькістю елементів керування програмою до елементів керування ОС Windows 10.

В програмі наявні три головні вкладки: Стан жорсткого диску, Пошук та виправлення "битих" секторів, Про програму.

Оскільки файл містить однакові (дублюючі) функції для елементів одного типу (кнопка, зображення, та ін.) для розуміння структури коду і функцій буде достатнім ознайомлення з частиною коду. Повний код наведено в додатку А. Пояснення наведено після коду.

```
private void InitializeComponent()
{
    System.ComponentModel.ComponentResourceManager resources = new
System.ComponentModel.ComponentResourceManager(typeof(MainForm));
    this.pictureBoxMap = new System.Windows.Forms.PictureBox();
    this.btnStart = new System.Windows.Forms.Button();
    this.comboDisks = new System.Windows.Forms.ComboBox();
    this.chkRead = new System.Windows.Forms.RadioButton();
    this.groupBox1 = new System.Windows.Forms.GroupBox();
    this.chkWriteVerify = new System.Windows.Forms.RadioButton();
    this.chkReadWriteVerifyRestore = new
System.Windows.Forms.RadioButton();
    this.chkReadRewriteVerify = new System.Windows.Forms.RadioButton();
    this.btnCopyLog = new System.Windows.Forms.Button();
    this.lblPosition = new System.Windows.Forms.Label();
    this.lblSpeed = new System.Windows.Forms.Label();
    this.groupProgress = new System.Windows.Forms.GroupBox();
    this.pictureBoxLegend = new System.Windows.Forms.PictureBox();
```

```

        this.groupBox2 = new System.Windows.Forms.GroupBox();
        this.tabControl1 = new System.Windows.Forms.TabControl();
        this.tabPage2 = new System.Windows.Forms.TabPage();
        this.listView1 = new System.Windows.Forms.ListView();
        this.columnHeader1 = ((System.Windows.Forms.ColumnHeader)(new
System.Windows.Forms.ColumnHeader()));
        this.columnHeader2 = ((System.Windows.Forms.ColumnHeader)(new
System.Windows.Forms.ColumnHeader()));
        this.columnHeader3 = ((System.Windows.Forms.ColumnHeader)(new
System.Windows.Forms.ColumnHeader()));
        this.columnHeader4 = ((System.Windows.Forms.ColumnHeader)(new
System.Windows.Forms.ColumnHeader()));
        this.columnHeader6 = ((System.Windows.Forms.ColumnHeader)(new
System.Windows.Forms.ColumnHeader()));
        this.button1 = new System.Windows.Forms.Button();
        this.comboSMARTDisks = new System.Windows.Forms.ComboBox();
        this.label6 = new System.Windows.Forms.Label();
        this.tabPage1 = new System.Windows.Forms.TabPage();
        this.label1 = new System.Windows.Forms.Label();
        this.tabPage3 = new System.Windows.Forms.TabPage();
        this.label3 = new System.Windows.Forms.Label();
        this.label2 = new System.Windows.Forms.Label();
        this.pictureBox2 = new System.Windows.Forms.PictureBox();
        this.pictureBox1 = new System.Windows.Forms.PictureBox();
        this.label4 = new System.Windows.Forms.Label();
        this.label5 = new System.Windows.Forms.Label();
        this.groupBox3 = new System.Windows.Forms.GroupBox();
        this.radioButton3 = new System.Windows.Forms.RadioButton();
        this.radioButton2 = new System.Windows.Forms.RadioButton();
        this.radioButton1 = new System.Windows.Forms.RadioButton();
        ((System.ComponentModel.ISupportInitialize)(this.pictureBoxMap)).BeginInit();
        this.groupBox1.SuspendLayout();
        this.groupProgress.SuspendLayout();
        ((System.ComponentModel.ISupportInitialize)(this.pictureBoxLegend)).BeginInit();
        this.groupBox2.SuspendLayout();
        this.tabControl1.SuspendLayout();
        this.tabPage2.SuspendLayout();
        this.tabPage1.SuspendLayout();
        this.tabPage3.SuspendLayout();
        ((System.ComponentModel.ISupportInitialize)(this.pictureBox2)).BeginInit();
        ((System.ComponentModel.ISupportInitialize)(this.pictureBox1)).BeginInit();
        this.groupBox3.SuspendLayout();
        this.SuspendLayout();
        //
        // pictureBoxMap
        //
        this.pictureBoxMap.Location = new System.Drawing.Point(19, 65);
        this.pictureBoxMap.Name = "pictureBoxMap";
        this.pictureBoxMap.Size = new System.Drawing.Size(609, 403);
        this.pictureBoxMap.TabIndex = 0;
        this.pictureBoxMap.TabStop = false;
        this.pictureBoxMap.Paint += new
System.Windows.Forms.PaintEventHandler(this.pictureBoxMap_Paint);
        //
        // btnStart
        //
        this.btnStart.Font = new System.Drawing.Font("Microsoft Sans Serif",
11.25F, System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point,
(byte)(204));
        this.btnStart.Location = new System.Drawing.Point(655, 144);
        this.btnStart.Name = "btnStart";
        this.btnStart.Size = new System.Drawing.Size(225, 37);
        this.btnStart.TabIndex = 6;

```

```

        this.btnStart.Text = "Старт";
        this.btnStart.UseVisualStyleBackColor = true;
        this.btnStart.Click += new System.EventHandler(this.btnStart_Click);
        //
        // comboDisks
        //
        this.comboDisks.DropDownStyle =
System.Windows.Forms.ComboBoxStyle.DropDownList;
        this.comboDisks.Enabled = false;
        this.comboDisks.Font = new System.Drawing.Font("Microsoft Sans
Serif", 11.25F, System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, ((byte) (204)));
        this.comboDisks.FormattingEnabled = true;
        this.comboDisks.Location = new System.Drawing.Point(151, 17);
        this.comboDisks.Name = "comboDisks";
        this.comboDisks.Size = new System.Drawing.Size(351, 26);
        this.comboDisks.TabIndex = 2;
        this.comboDisks.SelectedIndexChanged += new
System.EventHandler(this.comboDisks_SelectedIndexChanged);
        //
        // chkRead
        //
        this.chkRead.AutoSize = true;
        this.chkRead.Checked = true;
        this.chkRead.Font = new System.Drawing.Font("Microsoft Sans Serif",
11.25F, System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point,
((byte) (204)));
        this.chkRead.Location = new System.Drawing.Point(6, 20);
        this.chkRead.Name = "chkRead";
        this.chkRead.Size = new System.Drawing.Size(188, 22);
        this.chkRead.TabStop = true;
        this.chkRead.Text = "Пошук \"битих\" секторів";
        this.chkRead.UseVisualStyleBackColor = true;

```

Цей код ініціалізує елементи інтерфейсу форми і саму форму. Для кращого візуального сприйняття опис кожного елементу відокремлено наступною конструкцією з коментарів:

```

//
// name
//

```

Така конструкція є загальновживаним рішенням, яке застосовують з метою підвищення читабельності коду.

Кожен графічний елемент інтерфейсу (кнопка, зображення, текстова область та ін.) має наступні властивості:

- .Location це позиція елементу відносно форми;
- .Name це ім'я елементу в програмному коді;
- .Size це розмір елементу на формі;
- .Text це текстове повідомлення, яке міститься на елементі і відображається користувачеві.

В деяких графічних елементах наявні додаткові властивості, що враховують особливості елемента. Наприклад для елемента PictureBox, який зберігає і відтворює зображення наявна властивість Image, яка зберігає адресу зображення. Деякі властивості можуть бути відсутні – в такому разі для них буде застосовано значення за замовчуванням.

В результаті обробки файлів розмітки формується інтерфейс вікна. За результатами обробки MainForm.Designer.cs формується інтерфейс головної форми програми.

4 КЕРІВНИЦТВО ПРОГРАМІСТА

4.1 Призначення і умови застосування програми

Найменування продукту – програма «HDD Easy Scanner». Програма призначена для моніторингу стану ЖД.

Область застосування – отримання інформації про стан обладнання ПК.

Дана розробка дає можливість користувачу утримувати ЖД придатним для надійного зберігання інформації користувача та у робочому стані.

Вимоги до складу і параметрів технічних засобів – для функціонування системи необхідні технічні засоби з наступними параметрами:

- ПК з мікропроцесором Intel Pentium 1,0 ГГц та вище;
- наявність вільного місця на жорсткому диску 700 Кб (для програми) та 16 Гб для 32-розрядної ОС або 20 Гб для 64-розрядної ОС;
- оперативна пам'ять 1 Гб (для 32-розрядної версії) або 2 Гб (для 64-розрядної версії) ОС;
- відеоадаптер з DirectX 9 або новішої версії з драйвером WDDM 1.0;
- дисплей мінімум 800x600 пікселів;
- клавіатура і маніпулятор типу "миша".

Вимоги до інформаційної та програмної сумісності – для функціонування програми необхідна наявність ОС Windows, версії 10 з встановленим .NET Framework версії не нижче 4.6.1.

4.2 Основні вимоги до системи

Вимоги до функціональних характеристик програми «HDD Easy Scanner»:

- програма має реагувати на натискання клавіш миші та клавіатури з затримкою не більше 0.5 с;
- програма повинна надавати можливість обирати необхідний ЖД;

- програма повинна візуально відображати процес пошуку «битих» секторів;
- програма повинна надавати можливість вибору методу пошуку «битих» секторів;
- програма в момент пошуку «битих» секторів повинна сповіщати про поточну позицію пошуку та швидкість;
- програма повинна надавати можливість оновлювати значення S.M.A.R.T. атрибутів.

4.3 Характеристика програми

Найменування програми «HDD Easy Scanner».

Обсяг оперативної пам'яті, який займає програма – 700 Кб.

Програма запускається при необхідності моніторингу стану ЖД.

Перелік файлів та тек, в яких знаходиться програма:

- DiskAccessLibrary – тека з додатковою бібліотекою;
- Utilities – тека з додатковою бібліотекою;
- app.config – користувацькі налаштування програми;
- MainForm.cs – код-логіка головної форми;
- MainForm.Designer.cs – інтерфейс головної форми;
- MainForm.resx – ресурси (параметри) головної форми ;
- packages.config – перелік встановлених компонентів;
- Program.cs – параметри запуску програми;
- Smart.cs – код класу Smart;
- HDD.cs – код класу HDD;
- UIHelper.cs – код класу UIHelper;
- DiskReader.cs – код класу DiskReader;
- DiskTester.cs – код класу DiskTester;
- HardDisk.ico – іконка програми.

4.4 Початкові та вихідні дані

Початковими даними, необхідними для функціонування програмного продукту, є поточна інформація про стан ЖД, яка буде автоматично отримуватися програмою.

Вихідними даними є наглядне відображення інформації про стан жорсткого диску.

4.5 Повідомлення програмісту

Повідомлення програмісту не передбачені.

5 КЕРІВНИЦТВО ОПЕРАТОРА

5.1 Призначення програми

Програмне забезпечення «HDD Easy Scanner» призначене для перегляду значень S.M.A.R.T. атрибутів та проведення перевірки на наявність пошкоджених секторів обраного жорсткого диску.

Галузь використання – сервісне обслуговування комп'ютерних систем на етапі експлуатації.

Даний програмний продукт надає можливість користувачу утримувати ЖД у робочому стані та підвищити надійність зберігання інформації.

5.2 Умови використання програми

Вимоги до складу і параметрів технічних засобів зазначені у п.1.3.4 технічного завдання.

Вимоги до інформаційної та програмної сумісності наступні: для функціонування системи необхідна наявність ОС Windows, версії 10 з встановленим .NET Framework версії не нижче 4.6.1.

5.3 Виконання програми

Для початку роботи з застосунком необхідно запустити файл «HDD Easy Scanner.exe». Після запуску програми відкривається головна форма наведена, з позначками кнопок керування, на рис. 5.1.

Нижче наведено пояснення позначок.

Позначка 1 – Кнопка, що дозволить перейти до вкладки «Стан жорсткого диску».

Позначка 2 – Кнопка, що дозволить перейти до вкладки «Пошук та виправлення "битих" секторів».

Позначка 3 – Кнопка, що дозволить перейти до вкладки «Про програму».

Позначка 4 – Кнопки, що дозволяє обрати мову інтерфейсу.

Позначка 5 – Кнопка, яка дає можливість обрати необхідний жорсткий диск.

Позначка 6 – Кнопка, що оновлює значення S.M.A.R.T. атрибутів.

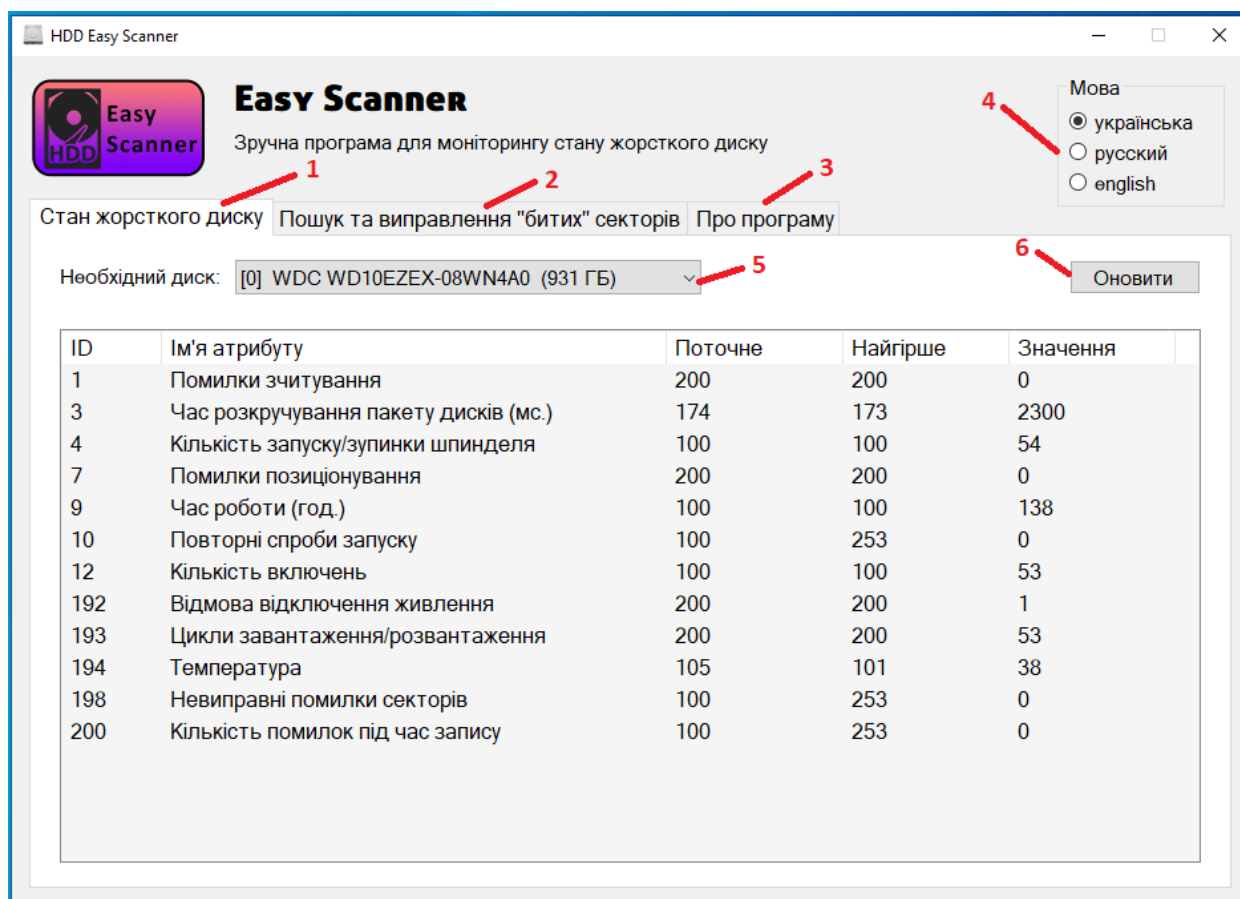


Рисунок 5.1 – Головна форма з позначками кнопок керування

При натисненні на кнопку з позначкою 2 відкриється друга вкладка наведена, з позначками кнопок керування, на рис. 5.2

Нижче наведено пояснення позначок.

Позначка 1 – Кнопка, що дозволить перейти до вкладки «Стан жорсткого диску».

Позначка 2 – Кнопка, що дозволить перейти до вкладки «Пошук та виправлення "битих" секторів».

Позначка 3 – Кнопка, що дозволить перейти до вкладки «Про програму».

Позначка 4 – Кнопки, що дозволяє обрати мову інтерфейсу.

Позначка 5 – Кнопка, яка дає можливість обрати необхідний жорсткий диск.

Позначка 6 – Кнопка, що дозволяє обрати метод перевірки жорсткого диску.

Позначка 7 – Кнопка, яка запускає перевірку з вибраним методом.

Позначка 8 – Кнопка, що дозволяє скопіювати лог виконання перевірки до буферу обміну.

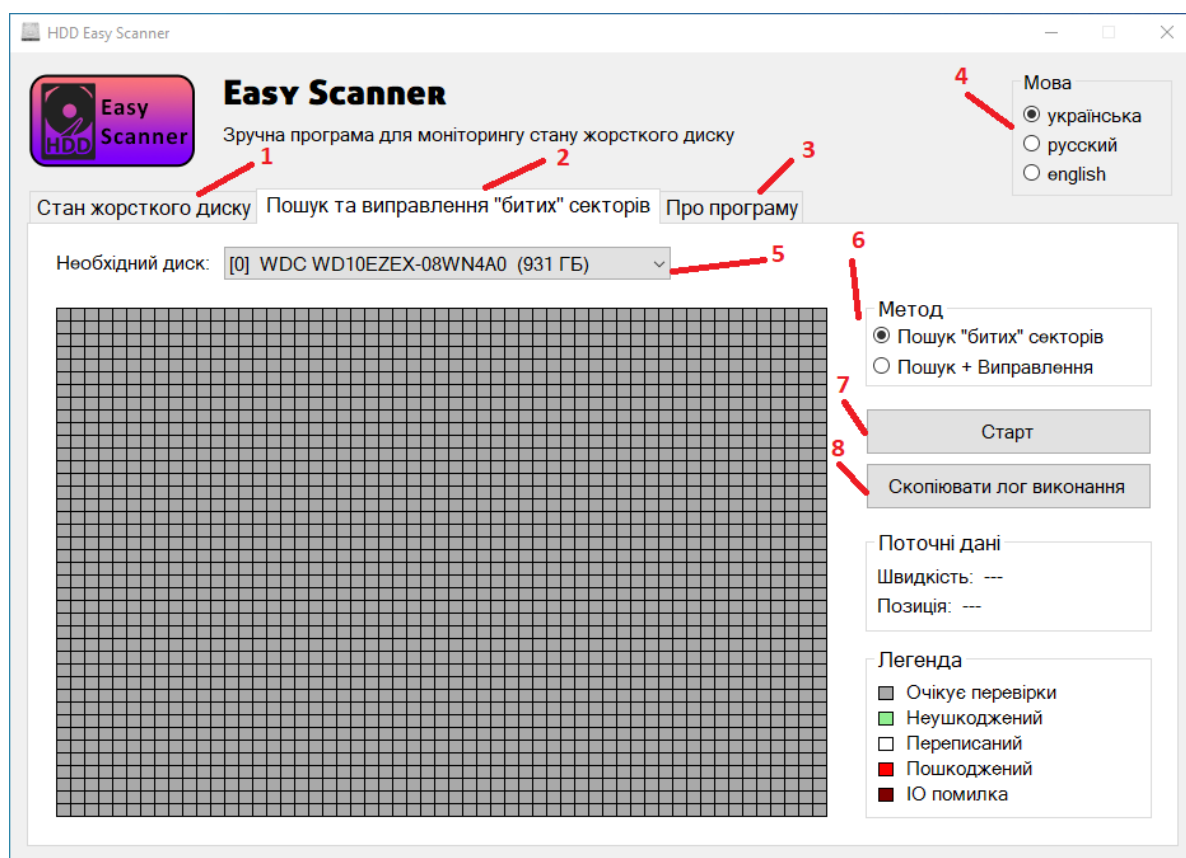


Рисунок 5.2 – Вкладка «Пошук та виправлення "битих" секторів» з позначками кнопок керування

Дані, які можна побачити в колонці «Значення», (рис. 5.1) надають інформацію про надійність щодо деякого еталонного значення. Зазвичай атрибути мають значення в діапазоні від 0 до 100, хоча деякі атрибути мають значення від 0 до 200 і від 0 до 253.

Висока оцінка говорить про відсутність змін параметра, що контролюється або повільне його погіршення, низька – про можливий швидкий збій.

Значення, менше за мінімальне значення, при якому виробником гарантується безвідмовна робота накопичувача, означає вихід вузла з ладу.

Опис S.M.A.R.T. атрибутів наведено у табл. 5.1.

Таблиця 5.1 – Опис S.M.A.R.T. атрибутів

№	Назва атрибуту	Опис
1	2	3
1	Помилки зчитування	Частота помилок при читанні даних з диска, походження яких зумовлено апаратною частиною диска. Чим нижче значення, тим краще.
3	Час розкручування пакету дисків, мс	Час розкрутки пакета дисків зі стану спокою до робочої швидкості. Ростає при зносі механіки (підвищене тертя). Також може свідчити про неякісне живлення. Чим нижче значення, тим краще.
4	Кількість запусків/зупинок шпинделя	Повне число циклів запуск-зупинка шпинделя. У дисків деяких виробників (наприклад, Seagate) - лічильник включення режиму енергозбереження. В полі «Значення» зберігається загальна кількість запусків/зупинок диска.
7	Помилки позиціонування	Частота помилок при позиціонуванні блоку магнітних головок. Чим їх більше, тим гірше стан механіки і/або поверхні жорсткого диска. Також на значення параметра може вплинути перегрів і зовнішні вібрації (наприклад, від сусідніх дисків).

Продовження таблиці 5.1

1	2	3
9	Час роботи, год.	Число годин, проведених у включеному стані.
10	Повторні спроби запуску	Число повторних спроб розкрутки дисків до робочої швидкості в разі, якщо перша спроба була невдалою. Якщо значення атрибута збільшується, то велика ймовірність неполадок з механічною частиною. Чим нижче значення, тим краще.
12	Кількість включень	Кількість повних циклів включення-виключення диска.
13	Кількість помилок при зчитуванні	Число помилок при читанні, з вини програмного забезпечення, які не піддалися виправленню. Всі помилки мають не механічну природу і вказують лише на неправильну розмітку / взаємодія з диском програм або операційної системи. Чим нижче значення, тим краще.
188	Кількість перерваних операцій через HDD тайм-аут	Кількість перерваних операцій в зв'язку з HDD тайм-аут. Зазвичай це значення атрибута має дорівнювати нулю, і, якщо значення набагато вище нуля, то, швидше за все, там будуть якісь серйозні проблеми з живленням. Чим нижче значення, тим краще.
189	Кількість випадків запису при висоті головки вище стандартної	Містить кількість зафіксованих випадків запису при висоті польоту головки вище розрахованої - швидше за все, через зовнішніх впливів, наприклад вібрації. Чим нижче значення, тим краще.

Продовження таблиці 5.1

1	2	3
191	Кількість помилок через ударні навантаження	Кількість помилок, що виникають в результаті ударних навантажень. Атрибут зберігає свідчення вбудованого акселерометра, який фіксує всі удари, поштовхи, падіння і навіть неакуратну установку диска в корпус комп'ютера. Чим нижче значення, тим краще.
192	Відмова відключення живлення	Число циклів виключень або аварійних відмов (включень/виключень живлення накопичувача). Чим нижче значення, тим краще.
193	Цикли завантаження/розвантаження	Кількість циклів переміщення блоку магнітних головок в паркувальну зону/ в робоче положення.
194	Температура	Тут зберігаються дані вбудованого термодатчика для механічної частини диска. Чим нижче значення, тим краще.
198	Невиправні помилки секторів	Число невиправних (засобами диска) секторів. У разі збільшення числа помилок, велика ймовірність критичних дефектів поверхні і/або механіки накопичувача. Чим нижче значення, тим краще.
200	Кількість помилок під час запису	Показує загальне число помилок запису на диск. Може служити показником якості поверхні і механіки накопичувача. Чим нижче значення, тим краще.

Продовження таблиці 5.1

1	2	3
207	Сила струму при розкручуванні диску	Величина сили струму при розкручуванні диска. Чим нижче значення, тим краще.
220	Струс диску	Дистанція зміщення блоку дисків щодо шпинделя. В основному виникає через удар або падіння. Одиниця виміру невідома. При збільшенні атрибута диск швидко стає непрацездатним.
223	Повтори завантаження/розвантаження	Кількість нових спроб вивантажень/завантажень блоку магнітних головок в/з паркувальну область після невдалої спроби.
226	Час завантаження головок, мс	Час, за який привід вивантажує магнітні головки з паркувальної області на робочу поверхню диска. Чим нижче значення, тим краще.
240	Час знаходження блоку головок у робочому стані, год.	Загальний час знаходження блоку головок в робочому положенні.
254	Захист від падіння	Показує, скільки раз диск падав. Чим нижче значення, тим краще.

Приклад результатів тестування ЖД наведено на рис. 5.3 та 5.4.

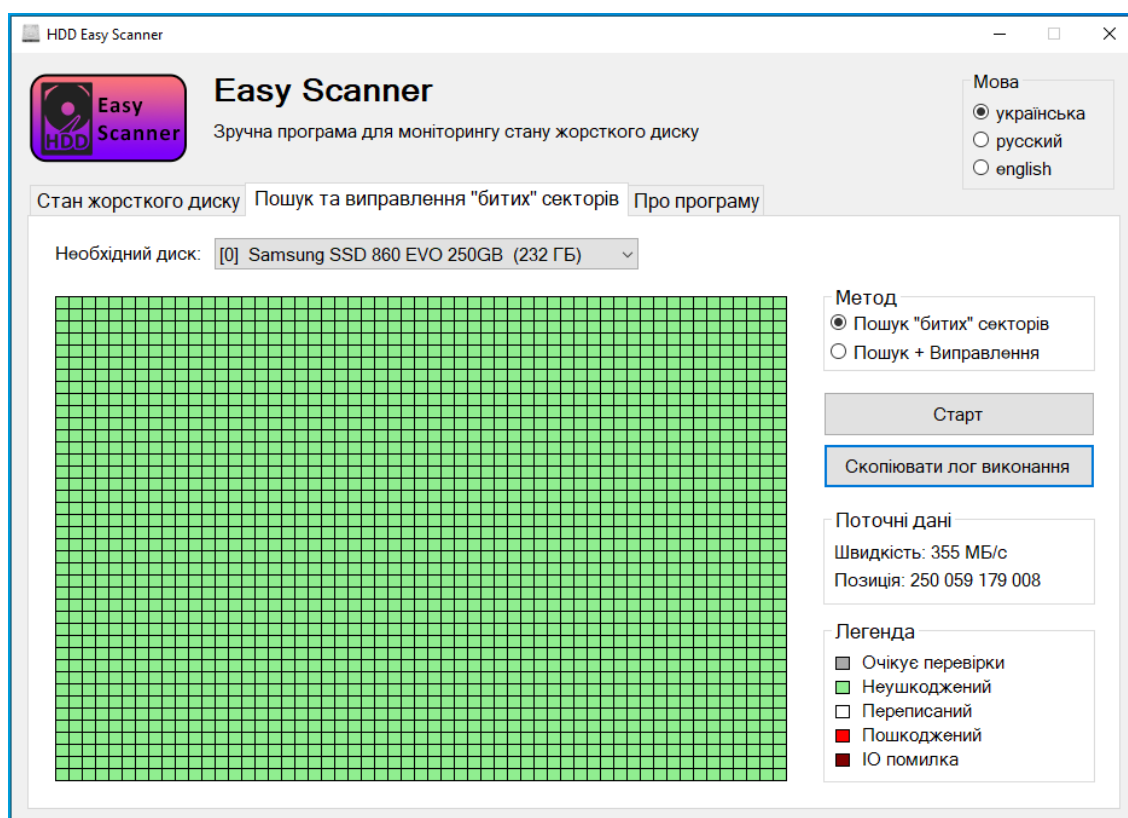


Рисунок 5.3 – Результати тестування на пошук «битих секторів»

2021-04-08 12:56:36: HDD Easy Scanner

2021-04-08 12:56:36: Starting Read Test

2021-04-08 12:56:36: Disk: Samsung SSD 860 EVO 250GB, S/N: S4G2NJ0N226459L

2021-04-08 12:56:36: Disk size: 232 ГБ (488 397 168 sectors, 512 bytes per sector)

2021-04-08 13:07:46: Test Completed|

Рисунок 5.4 – Лог виконання тестування на пошук «битих секторів»

При натисненні на кнопку з позначкою 3 (див. рис. 5.1) відкриється третя вкладка наведена на рис. 5.5



Рисунок 5.5 – Інтерфейс вкладки «Про програму»

5.4 Повідомлення оператору

В програмі передбачені два повідомлення оператору, які виникають у наступних випадках :

- при спробі скопіювати лог виконання програми;
- при спробі скопіювати порожній лог виконання програми.

При спробі скопіювати лог виконання програми за допомогою кнопки «Скопіювати лог виконання» (позначка 8 на рис. 5.2) оператору відкриється повідомлення з інформацією про успішне копіювання (рис. 5.6).

Якщо ж виявиться, що лог виконання порожній, то відкриється повідомлення про помилку копіювання (рис. 5.7).

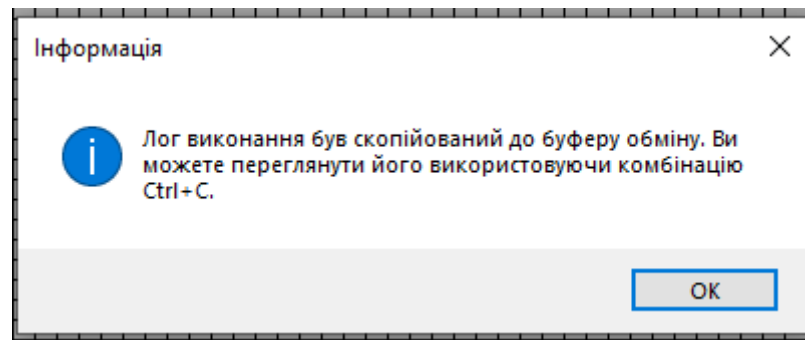


Рисунок 5.6 – Форма-повідомлення успішного копіювання логу виконання програми

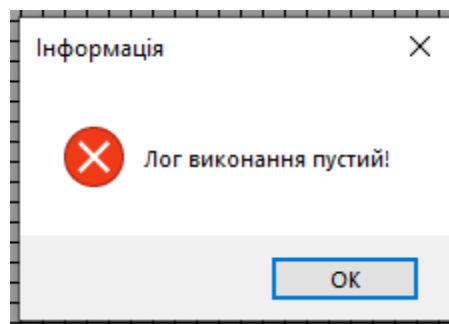


Рисунок 5.7 – Форма-повідомлення невдалого копіювання логу виконання програми

ВИСНОВКИ

У результаті виконання дипломної кваліфікаційної роботи бакалавра було проведено аналіз предметної області, огляд причин появи пошкоджених секторів ЖД, а також існуючих програмних засобів, що вирішують завдання моніторингу стану ЖД. З урахуванням результатів огляду розроблено технічне завдання з переліком вимог до функціональності та інтерфейсу розробленого застосунку.

Для проєктування застосунку було обрано мова програмування C#, яка відповідає основним вимогам та можливостям, що висувалися до мови програмування. Середовищем розробки обрано Visual Studio, що надає широкі можливості розробнику в проєктуванні коду, розробці дизайну інтерфейсу користувача, має значно спрощені деякі операції та високу адаптованість до ОС сімейства Windows.

Для моніторингу стану ЖД обрано технологію виявлення, моніторингу та обслуговування несправностей S.M.A.R.T. з використанням більш 20 атрибутів.

Створений проєкт відповідає меті розробки, всім вимогам, що були пред'явлені до програми в ТЗ: основним вимогам до програмного продукту (п. 1.3.3); безпомилково працює під керуванням ОС Windows, версії 10 з встановленим .NET Framework версії 4.6.1, відповідно до п. 1.3.4; супроводжується керівництвом програміста, керівництвом оператора, пояснювальною запискою на розроблений застосунок, вихідним кодом програми, відповідно до п. 1.3.7 та ін.

Застосунок надає можливість проведення перевірки ЖД на наявність пошкоджених секторів з візуалізацією результатів перевірки у вигляді карти, що відображає стан поверхні ЖД. Карта складається з 2200 візуальних блоків, кожен з яких показує інформацію про певну перевірену ділянку ЖД. Крім того користувач отримує інформацію про загальний час роботи та поточну температуру жорсткого диску. Аналіз цієї інформації, а також значень S.M.A.R.T. атрибутів дозволить підвищити надійність зберігання інформації користувача за рахунок її своєчасної перезапису або ж заміни носія.

Шляхи подальшого удосконалення застосунку наступні:

- впровадження можливості автоматичного завантаження з ОС;
- додання запланованого пошуку пошкоджених секторів через певні проміжки часу.

Галузь використання розробленого програмного забезпечення – сервісне обслуговування комп'ютерних систем на етапі експлуатації.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Ian, P. How to check and monitor your hard drive's health [Electronic resource] / P. Ian. – Access mode: <https://www.pcworld.com/article/3566868/how-to-check-and-monitor-your-hard-drives-health.html>.
2. Hard Drive Usage Monitoring Software [Electronic resource]. – Access mode: <https://www.dnsstuff.com/hard-drive-monitoring-software>.
3. Hoffman, C. Bad Sectors Explained: Why Hard Drives Get Bad Sectors and What You Can Do About It [Electronic resource] / C. Hoffman. – Access mode: <https://www.howtogeek.com/173463/bad-sectors-explained-why-hard-drives-get-bad-sectors-and-what-you-can-do-about-it/>.
4. Kumar, S. 6 Worst Reasons for Hard Disk Failure and Data Recovery Solutions [Electronic resource] / S. Kumar. – Access mode: <https://www.stellarinfo.com/blog/6-worst-reasons-of-hard-disk-failure/>.
5. 8 Freeware To Check & Repair SSD Hard Disk With Bad Sectors [Electronic resource]. – Access mode: <https://www.geckoandfly.com/21498/check-repair-hard-disk-bad-sector/>.
6. Crystal Disk Info [Electronic resource]. – Access mode: <https://crystalmark.info/en/software/crystaldiskinfo>.
7. HD Tune Pro [Electronic resource]. – Access mode: <https://www.hdtune.com>.
8. Self-Monitoring, Analysis and Reporting Technology (SMART) [Electronic resource]. – Access mode: <https://www.techopedia.com/definition/4537/self-monitoring-analysis-and-reporting-technology-smart>.
9. Self-Monitoring, Analysis and Reporting Technology (S.M.A.R.T.) [Electronic resource]. – Access mode: https://kb.sandisk.com/app/answers/detail/a_id/8516/~self-monitoring%2C-analysis-and-reporting-technology-%28s.m.a.r.t.%29.

10. Overview of .NET Framework [Electronic resource]. – Access mode: <https://docs.microsoft.com/en-us/dotnet/framework/get-started/overview>.

11. The Good and the Bad of .NET Framework Programming [Electronic resource]. – Access mode: <https://www.altexsoft.com/blog/engineering/the-good-and-the-bad-of-net-framework-programming/>.

12. Ramel D. Next Up for WinForms in .NET 6 Is High DPI Support [Electronic resource] / D. Ramel. – Access mode: <https://visualstudiomagazine.com/articles/2020/12/11/winforms-net6.aspx>.

13. Desktop Windows Version Market Share Worldwide - March 2021 [Electronic resource]. – Access mode: <https://gs.statcounter.com/os-version-market-share/windows/desktop/worldwide>.

14. Operating System Market Share [Electronic resource]. – Access mode: <https://netmarketshare.com/operating-system-market-share.aspx?options>.

15. What's New With Windows? See 6 Top New Features in the October 2020 Update [Electronic resource]. – Access mode: <https://www.pcmag.com/news/whats-new-with-windows-see-6-top-new-features-in-the-october-2020-update>.

16. Quibina, B. Jr. Desktop Application Programming Languages [Electronic resource] / B. Jr. Quibina. – Access mode: <https://flexisourceit.com.au/resources/blog/desktop-application-programming-languages/>.

17. Fauerbach, Ch. How Do I Choose the Right Programming Language for My Team? Languages [Electronic resource] / Ch. Fauerbach. – Access mode: <https://medium.com/capital-one-tech/lessons-from-the-leadership-trenches-how-do-i-choose-the-right-programming-language-for-my-team-36222aa78b38>.

18. Програма «Victoria» [Электрон. ресурс]. – Режим доступа: <https://hdd.by/victoria/>.

ДОДАТОК А
Текст програми

A.1 Код файла MainForm.cs

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.IO;
using System.Text;
using System.Threading;
using System.Windows.Forms;
using DiskAccessLibrary;
using Utilities;
using System.Management;

namespace HardDiskValidator
{
    public partial class MainForm : Form
    {
        public const int HorizontalBlocks = 55;
        public const int VerticalBlocks = 40;
        private DateTime m_startTime;
        private BlockStatus[] m_blocks = new BlockStatus[HorizontalBlocks *
VerticalBlocks];
        private bool m_isBusy = false;
        private DiskTester m_diskTester;
        private bool m_isClosing = false;
        private List<string> m_log = new List<string>();

        public MainForm()
        {
            InitializeComponent();
            //this.Text += " " +
System.Reflection.Assembly.GetExecutingAssembly().GetName().Version.ToString(3);
        }
        private void MainForm_Load(object sender, EventArgs e)
        {
            PopulateDiskList(new ComboBox[] { comboDisks, comboSMARTDisks });
            //UpdateSMART_Attr(0);
        }
        private void pictureBoxMap_Paint(object sender, PaintEventArgs e)
        {
            DrawDiskMap(e.Graphics);
        }
        private void pictureBoxLegend_Paint(object sender, PaintEventArgs e)
        {
            DrawLegend(e.Graphics);
        }
        private void DrawDiskMap(Graphics graphics)
        {
            const int BlockWidth = 10; // 6
            const int BlockHeight = 9; // 5
            Pen pen = new Pen(Color.Black, 1);
            int width = HorizontalBlocks * (BlockWidth + 1);
            int height = VerticalBlocks * (BlockHeight + 1);
            for (int index = 0; index <= HorizontalBlocks; index++)
            {
                int x = (BlockWidth + 1) * index;
                graphics.DrawLine(pen, x, 0, x, height);
            }
            for (int index = 0; index <= VerticalBlocks; index++)

```

```

        {
            int y = (BlockHeight + 1) * index;
            graphics.DrawLine(pen, 0, y, width, y);
        }
        for (int index = 0; index < m_blocks.Length; index++)
        {
            int hIndex = index % HorizontalBlocks;
            int vIndex = index / HorizontalBlocks;
            int x = 1 + hIndex * (1 + BlockWidth);
            int y = 1 + vIndex * (1 + BlockHeight);
            SolidBrush brush = new
SolidBrush(UIHelper.GetColor(m_blocks[index]));
            graphics.FillRectangle(brush, x, y, BlockWidth, BlockHeight);
        }
    }
    private void DrawLegend(Graphics graphics)
    {
        DrawLegendEntry(graphics, 1, 10, BlockStatus.Untested, "Ï÷³êó°
iãðã³ðêè");
        DrawLegendEntry(graphics, 1, 30, BlockStatus.OK, "Íåøêîäæåîëé");
        DrawLegendEntry(graphics, 1, 50, BlockStatus.OverwriteOK,
"Ïäðåîëëñåîëé");
        DrawLegendEntry(graphics, 1, 70, BlockStatus.Damaged, "Ïîêîäæåîëé");
        DrawLegendEntry(graphics, 1, 90, BlockStatus.IOError, "IO îîëëëå");
    }
    private void DrawLegendEntry(Graphics graphics, float x, float y,
BlockStatus status, string text)
    {
        const int BlockWidth = 10;
        const int BlockHeight = 9;
        Pen pen = new Pen(Color.Black, 1);
        SolidBrush brush = new SolidBrush(UIHelper.GetColor(status));
        graphics.DrawRectangle(pen, x, y, BlockWidth + 1, BlockHeight + 1);
        graphics.FillRectangle(brush, x + 1, y + 1, BlockWidth, BlockHeight);
        Font font = new Font(FontFamily.GenericSansSerif, 11);
        graphics.DrawString(text, font, Brushes.Black, x + 20, y - 5);
    }
    private void comboDisks_SelectedIndexChanged(object sender, EventArgs e)
    {
        //int physicalDiskIndex = ((KeyValuePair<int,
string>)comboDisks.SelectedItem).Key;
        //PhysicalDisk disk = new PhysicalDisk(physicalDiskIndex);
        //lblSerialNumber.Text = "Ñãð³éíëë îîåð: " + disk.SerialNumber;
    }
    private void MainForm_FormClosing(object sender, FormClosingEventArgs e)
    {
        if (m_isBusy)
        {
            e.Cancel = true;
            m_diskTester.Abort = true;
            m_isClosing = true;
        }
    }
    private void btnCopyLog_Click(object sender, EventArgs e)
    {
        if (m_log.Count > 0)
        {
            StringBuilder builder = new StringBuilder();
            foreach (string message in m_log)
            {
                builder.AppendLine(message);
            }
            Clipboard.SetText(builder.ToString());
        }
    }

```

```

    }
}
private void PopulateDiskList(ComboBox[] comboBoxes)
{
    Thread thread = new Thread(delegate()
    {
        List<PhysicalDisk> disks = PhysicalDiskHelper.GetPhysicalDisks();
        this.Invoke((MethodInvoker)delegate
        {
            foreach (ComboBox comboBox in comboBoxes)
            {
                comboBox.Items.Clear();
                comboBox.DisplayMember = "Value";
                comboBox.ValueMember = "Key";
                foreach (PhysicalDisk disk in disks)
                {
                    string title = String.Format("[{0}] {1} ({2})",
disk.PhysicalDiskIndex, disk.Description, UIHelper.GetSizeString(disk.Size));
                    comboBox.Items.Add(new KeyValuePair<int,
string>(disk.PhysicalDiskIndex, title));
                }
                comboBox.SelectedIndex = 0;
                comboBox.Enabled = true;
            }
        });
    });
    thread.Start();
}
private void btnStart_Click(object sender, EventArgs e)
{
    if (m_isBusy)
    {
        btnStart.Enabled = false;
        btnStart.Text = "Çóíèíêà";
        Thread thread = new Thread(delegate()
        {
            m_diskTester.Abort = true;
            while (m_isBusy)
            {
                Thread.Sleep(100);
            }
            if (!m_isClosing)
            {
                this.Invoke((MethodInvoker)delegate
                {
                    btnStart.Text = "Ñòàðò";
                    btnStart.Enabled = true;
                    comboDisks.Enabled = true;
                    chkRead.Enabled = true;
                    chkReadRewriteVerify.Enabled = true;
                    chkReadWriteVerifyRestore.Enabled = true;
                    chkWriteVerify.Enabled = true;
                });
            }
        });
        thread.Start();
    }
    else
    {
        if (comboDisks.SelectedItem == null)
        {
            return;
        }
    }
}

```

```

        int physicalDiskIndex = ((KeyValuePair<int,
string>)comboDisks.SelectedItem).Key;
        TestName testName = TestName.Read;
        if (chkReadRewriteVerify.Checked)
        {
            testName = TestName.ReadWipeDamagedRead;
        }
        else if (chkReadWriteVerifyRestore.Checked)
        {
            testName = TestName.ReadWriteVerifyRestore;
        }
        else if (chkWriteVerify.Checked)
        {
            if ((Control.ModifierKeys & (Keys.Control | Keys.Shift)) ==
Keys.Control)
            {
                testName = TestName.Verify;
            }
            else
            {
                if ((Control.ModifierKeys & (Keys.Control | Keys.Shift))
== (Keys.Control | Keys.Shift))
                {
                    testName = TestName.Write;
                }
                else
                {
                    testName = TestName.WriteVerify;
                }
                DialogResult dialogResult = MessageBox.Show("This test
will erase all existing data on the selected disk, Are you sure?", "Warning",
MessageBoxButtons.YesNo);
                if (dialogResult == DialogResult.No)
                {
                    return;
                }
            }
        }
    }
    btnStart.Text = "Ñòìì";
    comboDisks.Enabled = false;
    chkRead.Enabled = false;
    chkReadRewriteVerify.Enabled = false;
    chkReadWriteVerifyRestore.Enabled = false;
    chkWriteVerify.Enabled = false;
    Thread thread = new Thread(delegate()
    {
        m_isBusy = true;
        PerformTest(physicalDiskIndex, testName);
        m_isBusy = false;
        if (m_isClosing)
        {
            this.Invoke((MethodInvoker)delegate
            {
                this.Close();
            });
        }
        else
        {
            this.Invoke((MethodInvoker)delegate
            {
                btnStart.Text = "Ñòàðò";
                comboDisks.Enabled = true;
                chkRead.Enabled = true;
            });
        }
    });

```

```

        chkReadRewriteVerify.Enabled = true;
        chkReadWriteVerifyRestore.Enabled = true;
        chkWriteVerify.Enabled = true;
    });
}
});
thread.Start();
}
}
private void PerformTest(int physicalDiskIndex, TestName testName)
{
    PhysicalDisk disk = new PhysicalDisk(physicalDiskIndex);
    m_startTime = DateTime.Now;
    ClearLog();
    //AddToLog("HDD          Easy          Scanner          {0}",
System.Reflection.Assembly.GetExecutingAssembly().GetName().Version.ToString(3))
;

    AddToLog("HDD Easy Scanner");
    AddToLog("Starting {0} Test", GetTestTitle(testName));
    AddToLog("Disk: {0}, S/N: {1}", disk.Description, disk.SerialNumber);
    AddToLog("Disk size: {0} ({1:###,###,###,###,##0} sectors, {2} bytes
per sector)", UIHelper.GetSizeString(disk.Size), disk.TotalSectors,
disk.BytesPerSector);
    if (!(testName == TestName.Read || testName == TestName.Verify))
    {
        bool success = disk.ExclusiveLock();
        if (!success)
        {
            MessageBox.Show("Failed to lock the disk.");
            return;
        }
        if (Environment.OSVersion.Version.Major >= 6)
        {
            success = disk.SetOnlineStatus(false, false);
            if (!success)
            {
                disk.ReleaseLock();
                MessageBox.Show("Failed to take the disk offline.");
                return;
            }
        }
    }
    m_diskTester = new DiskTester(testName, disk);
    m_diskTester.OnStatusUpdate += delegate(long currentPosition)
    {
        UpdateStatus(currentPosition);
    };
    m_diskTester.OnLogUpdate += delegate(string format, object[] args)
    {
        AddToLog(format, args);
    };
    // The last segment might be bigger than the others
    long uiBlockSize = disk.TotalSectors / (HorizontalBlocks *
VerticalBlocks);
    m_blocks = new BlockStatus[HorizontalBlocks * VerticalBlocks];
    this.Invoke((MethodInvoker) delegate
    {
        pictureBoxMap.Invalidate();
        pictureBoxMap.Update();
    });
    for (int uiBlockIndex = 0; uiBlockIndex < m_blocks.Length;
uiBlockIndex++)
    {

```

```

        long sectorIndex = uiBlockIndex * uiBlockSize;
        long sectorCount = uiBlockSize;
        if (uiBlockIndex == m_blocks.Length - 1)
        {
            sectorCount = disk.TotalSectors - ((m_blocks.Length - 1) *
uiBlockSize);
        }
        BlockStatus blockStatus = m_diskTester.PerformTest(sectorIndex,
sectorCount);
        m_blocks[uiBlockIndex] = blockStatus;
        this.Invoke((MethodInvoker)delegate
        {
            pictureBoxMap.Invalidate();
            pictureBoxMap.Update();
        });
        if (m_diskTester.Abort)
        {
            break;
        }
    }

    if (testName != TestName.Read)
    {
        if (Environment.OSVersion.Version.Major >= 6)
        {
            disk.SetOnlineStatus(true, false);
        }
        disk.ReleaseLock();
        disk.UpdateProperties();
    }
    if (m_diskTester.Abort)
    {
        AddToLog("Test Aborted");
    }
    else
    {
        AddToLog("Test Completed");
    }
    m_diskTester = null;
}
private void UpdateStatus(long currentPosition)
{
    if (InvokeRequired)
    {
        this.Invoke((MethodInvoker)delegate
        {
            UpdateStatus(currentPosition);
        });
    }
    else
    {
        int totalSeconds = (int)((TimeSpan)(DateTime.Now -
m_startTime)).TotalSeconds;
        totalSeconds = Math.Max(totalSeconds, 1);
        long speed = currentPosition / totalSeconds;
        lblSpeed.Text = String.Format("Øâëä³ñòù: {0}/c",
UIHelper.GetSizeString(speed));
        lblPosition.Text = String.Format("Ïîçèö³ÿ:
{0:###,###,###,###,##0}", currentPosition);
    }
}
private void MainForm_KeyDown(object sender, KeyEventArgs e)
{

```

```

        if (e.Control)
        {
            if (e.Shift)
            {
                chkWriteVerify.Text = GetTestTitle(TestName.Write);
            }
            else
            {
                chkWriteVerify.Text = GetTestTitle(TestName.Verify);
            }
        }
    }
}
private void MainForm_KeyUp(object sender, KeyEventArgs e)
{
    if (e.Control)
    {
        chkWriteVerify.Text = GetTestTitle(TestName.Verify);
    }
    else
    {
        chkWriteVerify.Text = GetTestTitle(TestName.WriteVerify);
    }
}
private void MainForm_Deactivate(object sender, EventArgs e)
{
    chkWriteVerify.Text = GetTestTitle(TestName.WriteVerify);
}
private void ClearLog()
{
    m_log = new List<string>();
}
private void AddToLog(string format, params object[] args)
{
    string message = String.Format(format, args);
    AddToLog(message);
}
private void AddToLog(string message)
{
    string messageFormatted = String.Format("{0}: {1}",
DateTime.Now.ToString("yyyy-MM-dd HH:mm:ss"), message);
    m_log.Add(messageFormatted);
}
private static string GetTestTitle(TestName testName)
{
    if (testName == TestName.Read)
    {
        return "Read";
    }
    else if (testName == TestName.ReadWipeDamagedRead)
    {
        return "Read + Wipe Damaged + Read";
    }
    else if (testName == TestName.ReadWriteVerifyRestore)
    {
        return "Read + Write + Verify + Restore";
    }
    else if (testName == TestName.WriteVerify)
    {
        return "Write + Verify";
    }
    else if (testName == TestName.Write)
    {
        return "Write";
    }
}

```

```

    }
    else
    {
        return "Verify";
    }
}
private void UpdateSMART()
{
    try
    {
        // retrieve list of drives on computer (this will return both
        HDD's and CDROM's and Virtual CDROM's)
        var dicDrives = new Dictionary<int, HDD>();
        var wdSearcher = new ManagementObjectSearcher("SELECT * FROM
Win32_DiskDrive");
        // extract model and interface information
        int iDriveIndex = 0;
        foreach (ManagementObject drive in wdSearcher.Get())
        {
            var hdd = new HDD();
            hdd.Model = drive["Model"].ToString().Trim();
            hdd.Type = drive["InterfaceType"].ToString().Trim();
            dicDrives.Add(iDriveIndex, hdd);
            iDriveIndex++;
        }
        var pmsearcher = new ManagementObjectSearcher("SELECT * FROM
Win32_PhysicalMedia");
        // retrieve hdd serial number
        iDriveIndex = 0;
        foreach (ManagementObject drive in pmsearcher.Get())
        {
            // because all physical media will be returned we need to exit
            // after the hard drives serial info is extracted
            if (iDriveIndex >= dicDrives.Count)
                break;
            dicDrives[iDriveIndex].Serial = drive["SerialNumber"] == null
? "None" : drive["SerialNumber"].ToString().Trim();
            iDriveIndex++;
        }
        // get wmi access to hdd
        var searcher = new ManagementObjectSearcher("Select * from
Win32_DiskDrive");
        searcher.Scope = new ManagementScope(@"\root\wmi");
        // check if SMART reports the drive is failing
        searcher.Query = new ObjectQuery("Select * from
MSStorageDriver_FailurePredictStatus");
        iDriveIndex = 0;
        foreach (ManagementObject drive in searcher.Get())
        {
            dicDrives[iDriveIndex].IsOK =
(bool)drive.Properties["PredictFailure"].Value == false;
            iDriveIndex++;
        }
        // retrieve attribute flags, value worste and vendor data
        information
        searcher.Query = new ObjectQuery("Select * from
MSStorageDriver_FailurePredictData");
        iDriveIndex = 0;
        foreach (ManagementObject data in searcher.Get())
        {
            Byte[] bytes =
(Byte[])data.Properties["VendorSpecific"].Value;
            for (int i = 0; i < 30; ++i)

```

```

        {
            try
            {
                int id = bytes[i * 12 + 2];
                int flags = bytes[i * 12 + 4]; // least significant
status byte, +3 most significant byte, but not used so ignored.
                //bool advisory =
                (flags & 0x1) == 0x0;

                bool failureImminent = (flags & 0x1) == 0x1;
                //bool onlineDataCollection = (flags & 0x2) == 0x2;
                int value = bytes[i * 12 + 5];
                int worst = bytes[i * 12 + 6];
                int vendordata = BitConverter.ToInt32(bytes, i * 12 +
7);

                if (id == 0) continue;
                var attr = dicDrives[iDriveIndex].Attributes[id];
                attr.Current = value;
                attr.Worst = worst;
                attr.Data = vendordata;
                attr.IsOK = failureImminent == false;
            }
            catch
            {
                // given key does not exist in attribute collection
(attribute not in the dictionary of attributes)
            }
        }
        iDriveIndex++;
    }
    // retrieve threshold values foreach attribute
    searcher.Query = new ObjectQuery("Select * from
MSStorageDriver_FailurePredictThresholds");
    iDriveIndex = 0;
    foreach (ManagementObject data in searcher.Get())
    {
        Byte[] bytes =
(Byte[])data.Properties["VendorSpecific"].Value;
        for (int i = 0; i < 30; ++i)
        {
            try
            {
                int id = bytes[i * 12 + 2];
                int thresh = bytes[i * 12 + 3];
                if (id == 0) continue;
                var attr = dicDrives[iDriveIndex].Attributes[id];
                attr.Threshold = thresh;
            }
            catch
            {
                // given key does not exist in attribute collection
(attribute not in the dictionary of attributes)
            }
        }
        iDriveIndex++;
    }
    // print
    listView1.Items.Clear();
    foreach (var drive in dicDrives)
    {
        foreach (var attr in drive.Value.Attributes)
        {
            if (attr.Value.HasData)
            {

```

```

        ListViewItem item = new
ListViewItem(attr.Value.ID.ToString());
        item.SubItems.Add(attr.Value.Attribute);
        item.SubItems.Add(attr.Value.Current.ToString());
        item.SubItems.Add(attr.Value.Worst.ToString());
        item.SubItems.Add(attr.Value.Threshold.ToString());
        item.SubItems.Add(attr.Value.Data.ToString());
        listView1.Items.Add(item);
    }
}
}
}
catch (ManagementException e)
{
}
}
private void comboSMARTDisks_SelectedIndexChanged(object sender,
EventArgs e)
{
    int physicalDiskIndex = ((KeyValuePair<int,
string>)comboSMARTDisks.SelectedItem).Key;
    UpdateSMART_Attr(physicalDiskIndex);
}
private void UpdateSMART_Attr(int Disk)
{
    listView1.Items.Clear();
    var searcher = new ManagementObjectSearcher("Select * from
Win32_DiskDrive");
    searcher.Scope = new ManagementScope(@"\root\wmi");
    HDD disk = new HDD();
    int counter = 0;
    // retrieve threshold values foreach attribute
    searcher.Query = new ObjectQuery("Select * from
MSStorageDriver_FailurePredictThresholds");
    foreach (ManagementObject data in searcher.Get())
    {
        if (counter++ != Disk)
            continue;
        Byte[] bytes = (Byte[])data.Properties["VendorSpecific"].Value;
        for (int i = 0; i < 30; ++i)
        {
            try
            {
                int id = bytes[i * 12 + 2];
                int thresh = bytes[i * 12 + 3];
                if (id == 0) continue;
                var attr = disk.Attributes[id];
                attr.Threshold = thresh;
            }
            catch
            {
                // given key does not exist in attribute collection
                (attribute not in the dictionary of attributes)
            }
        }
    }
    searcher.Query = new ObjectQuery("Select * from
MSStorageDriver_FailurePredictData");
    counter = 0;
    foreach (ManagementObject data in searcher.Get())
    {
        if (counter++ != Disk)

```

```

        continue;
Byte[] bytes = (Byte[])data.Properties["VendorSpecific"].Value;
for (int i = 0; i < 30; ++i)
{
    try
    {
        int id = bytes[i * 12 + 2];
        int flags = bytes[i * 12 + 4]; // least significant status
byte, +3 most significant byte, but not used so ignored.
//bool advisory = (flags &
0x1) == 0x0;

        bool failureImminent = (flags & 0x1) == 0x1;
        //bool onlineDataCollection = (flags & 0x2) == 0x2;
        int value = bytes[i * 12 + 5];
        int worst = bytes[i * 12 + 6];
        int vendordata = BitConverter.ToInt32(bytes, i * 12 + 7);
        if (id == 0) continue;
        var attr = disk.Attributes[id];
        attr.Current = value;
        attr.Worst = worst;
        attr.Data = vendordata;
        attr.IsOK = failureImminent == false;
        if (attr.HasData)
        {
            ListViewItem item = new
ListViewItem(attr.ID.ToString());
            item.SubItems.Add(attr.Attribute);
            item.SubItems.Add(attr.Current.ToString());
            item.SubItems.Add(attr.Worst.ToString());
            //item.SubItems.Add(attr.Threshold.ToString());
            item.SubItems.Add(attr.Data.ToString());
            listView1.Items.Add(item);
        }
    }
    catch
    {
        // given key does not exist in attribute collection
(attribute not in the dictionary of attributes)
    }
}
}
}
private void btnRefresh_Click(object sender, EventArgs e)
{
    int physicalDiskIndex = ((KeyValuePair<int,
string>)comboSMARTDisks.SelectedItem).Key;
    UpdateSMART_Attr(physicalDiskIndex);
}
}
}

```

A.2 Код файла Smart.cs

```

namespace HardDiskValidator
{
    public class Smart
    {
        public bool HasData
        {
            get

```

```

        {
            if (Current == 0 && Worst == 0 && Threshold == 0 && Data == 0)
                return false;
            return true;
        }
    }
    public int ID { get; set; }
    public string Attribute { get; set; }
    public int Current { get; set; }
    public int Worst { get; set; }
    public int Threshold { get; set; }
    public int Data { get; set; }
    public bool IsOK { get; set; }
    public Smart()
    {
    }
    public Smart(int id, string attributeName)
    {
        this.ID = id;
        this.Attribute = attributeName;
    }
}
}

```

A.3 Код файлу HDD.cs

```

using System.Collections.Generic;
namespace HardDiskValidator
{
    public class HDD
    {
        public int Index { get; set; }
        public bool IsOK { get; set; }
        public string Model { get; set; }
        public string Type { get; set; }
        public string Serial { get; set; }
        public Dictionary<int, Smart> Attributes = new Dictionary<int, Smart>() {
            {0x01, new Smart(0x01, "Помилки зчитування")},
            {0x03, new Smart(0x03, "Час розкручування пакету дисків (мс.)"}},
            {0x04, new Smart(0x04, "Кількість запуску/зупинки шпинделя")},
            {0x07, new Smart(0x07, "Помилки позиціонування")},
            {0x09, new Smart(0x09, "Час роботи (год.)"}},
            {0x0A, new Smart(0x0A, "Повторні спроби запуску")},
            {0x0C, new Smart(0x0C, "Кількість включень")},
            {0x0D, new Smart(0x0D, "Кількість помилок при зчитуванні")},
            {0xBC, new Smart(0xBC, "Кількість перерваних операцій через HDD тайм-
аут")},
            {0xBD, new Smart(0xBD, "Кількість випадків запису при висоті головки
вище стандартної")},
            {0xBF, new Smart(0xBF, "Кількість помилок через ударні навантаження")},
            {0xC0, new Smart(0xC0, "Відмова відключення живлення")},
            {0xC1, new Smart(0xC1, "Цикли завантаження/розвантаження")},
            {0xC2, new Smart(0xC2, "Температура")},
            {0xC6, new Smart(0xC6, "Невиправні помилки секторів")},
            {0xC8, new Smart(0xC8, "Кількість помилок під час запису")},
            {0xCF, new Smart(0xCF, "Сила струму при розкручуванні диску")},
            {0xDC, new Smart(0xDC, "Струс диску")},
            {0xDF, new Smart(0xDF, "Повтори завантаження/розвантаження")},
            {0xE2, new Smart(0xE2, "Час завантаження головок (мс.)"}},
        };
    }
}

```

```

        {0xF0, new Smart(0xF0, "Час знаходження блока головок у робочому стані
(год.)")}},
        {0xFE, new Smart(0xFE, "Захист від падіння")}},
    };
}
}

```

A.4 Код файлу UIHelper.cs

```

using System;
using System.Collections.Generic;
using System.Drawing;
using System.Text;

namespace HardDiskValidator
{
    public class UIHelper
    {
        public static Color GetColor(BlockStatus status)
        {
            switch (status)
            {
                case BlockStatus.Untested:
                    return Color.DarkGray;
                case BlockStatus.OK:
                    return Color.LightGreen;
                case BlockStatus.OverwriteOK:
                    return Color.White;
                case BlockStatus.Damaged:
                    return Color.Red;
                default:
                    return Color.Maroon;
            }
        }

        public static string GetSizeString(long value)
        {
            string[] suffixes = { "Б", "КБ", "МБ", "ГБ", "ТБ", "ПБ", "ЕБ" };
            int suffixIndex = 0;
            while (value > 9999 && suffixIndex < suffixes.Length - 1)
            {
                value = value / 1024;
                suffixIndex++;
            }
            return String.Format("{0} {1}", value.ToString(),
suffixes[suffixIndex]);
        }
    }
}

```

A.5 Код файлу DiskReader.cs

```

using System;
using System.Collections.Generic;
using System.IO;
using DiskAccessLibrary;
using Utilities;

namespace HardDiskValidator

```

```

{
    public class DiskReader
    {
        private Disk m_disk;
        private bool m_abort;
        public delegate void UpdateStatusHandler(long currentPosition);
        public event UpdateStatusHandler OnStatusUpdate;
        public delegate void AddToLogHandler(string format, params object[] args);
        public event AddToLogHandler OnLogUpdate;
        public DiskReader(Disk disk)
        {
            m_disk = disk;
        }
        /// <returns>Will return null if an error has occurred or read is
        aborted</returns>
        public byte[] ReadSectors(long sectorIndex, long sectorCount, out bool
        ioErrorOccurred)
        {
            ioErrorOccurred = false;
            if (sectorCount > PhysicalDisk.MaximumDirectTransferSizeLBA)
            {
                // We must read one segment at a time, and copy the segments to a
                big buffer
                byte[] buffer = new byte[sectorCount * m_disk.BytesPerSector];
                for (long sectorOffset = 0; sectorOffset < sectorCount;
                sectorOffset += PhysicalDisk.MaximumDirectTransferSizeLBA)
                {
                    long leftToRead = sectorCount - sectorOffset;
                    int sectorsToRead = (int)Math.Min(leftToRead,
                    PhysicalDisk.MaximumDirectTransferSizeLBA);
                    byte[] segment = ReadSectorsUnbuffered(sectorIndex +
                    sectorOffset, sectorsToRead, out ioErrorOccurred);
                    if (m_abort || ioErrorOccurred || segment == null)
                    {
                        long nextOffset = sectorOffset + sectorsToRead;
                        if ((ioErrorOccurred || segment == null) && (sectorCount -
                        nextOffset > 0))
                        {
                            AddToLog("Skipped {0:###,###,###,###,##0}-
                            {1:###,###,###,###,##0}", sectorIndex + nextOffset, sectorIndex + sectorCount -
                            1);
                        }
                        return null;
                    }
                    Array.Copy(segment, 0, buffer, sectorOffset *
                    m_disk.BytesPerSector, segment.Length);
                }
                return buffer;
            }
            else
            {
                return ReadSectorsUnbuffered(sectorIndex, (int)sectorCount, out
                ioErrorOccurred);
            }
        }
        /// <returns>Will return null if an error has occurred</returns>
        public byte[] ReadSectorsUnbuffered(long sectorIndex, int sectorCount,
        out bool ioErrorOccurred)
        {
            ioErrorOccurred = false;
            UpdateStatus(sectorIndex * m_disk.BytesPerSector);
            try
            {

```

```

        return m_disk.ReadSectors(sectorIndex, sectorCount);
    }
    catch (IOException ex)
    {
        int errorCode = IOExceptionHelper.GetWin32ErrorCode(ex);
        AddToLog("Read failure (Win32 error: {0}) at {1:###,###,###,###,##0}-{2:###,###,###,###,##0}", errorCode, sectorIndex,
sectorIndex + sectorCount - 1);
        if (errorCode != (int)Win32Error.ERROR_IO_DEVICE && errorCode !=
(int)Win32Error.ERROR_CRC)
        {
            ioErrorOccured = true;
        }
        return null;
    }
}
/// <returns>Will return null if an unrecoverable IO Error has occurred or
read is aborted</returns>
public byte[] ReadEverySector(long sectorIndex, int sectorCount, out
List<long> damagedSectors, out bool ioErrorOccured)
{
    if (sectorCount > PhysicalDisk.MaximumDirectTransferSizeLBA)
    {
        damagedSectors = new List<long>();
        ioErrorOccured = false;
        // we must read one segment at the time, and copy the segments to
a big buffer
        byte[] buffer = new byte[sectorCount * m_disk.BytesPerSector];
        for (int sectorOffset = 0; sectorOffset < sectorCount;
sectorOffset += PhysicalDisk.MaximumDirectTransferSizeLBA)
        {
            int leftToRead = sectorCount - sectorOffset;
            int sectorsToRead = (int)Math.Min(leftToRead,
PhysicalDisk.MaximumDirectTransferSizeLBA);
            List<long> damagedSectorsInSegment;
            bool ioErrorOccuredInSegment;
            byte[] segment = ReadEverySectorUnbuffered(sectorIndex +
sectorOffset, sectorsToRead, out damagedSectorsInSegment, out
ioErrorOccuredInSegment);
            damagedSectors.AddRange(damagedSectorsInSegment);
            if (ioErrorOccuredInSegment)
            {
                ioErrorOccured = true;
                return null;
            }
            if (m_abort)
            {
                return null;
            }
            Array.Copy(segment, 0, buffer, sectorOffset *
m_disk.BytesPerSector, segment.Length);
        }
        return buffer;
    }
    else
    {
        return ReadEverySectorUnbuffered(sectorIndex, sectorCount, out
damagedSectors, out ioErrorOccured);
    }
}
/// <returns>Will return null if an unrecoverable IO Error has occurred or
read is aborted</returns>

```

```

    public byte[] ReadEverySectorUnbuffered(long sectorIndex, int
sectorCount, out List<long> damagedSectors, out bool ioErrorOccured)
    {
        damagedSectors = new List<long>();
        ioErrorOccured = false;
        try
        {
            UpdateStatus(sectorIndex * m_disk.BytesPerSector);
            return m_disk.ReadSectors(sectorIndex, sectorCount);
        }
        catch (IOException ex1)
        {
            int errorCode1 = IOExceptionHelper.GetWin32ErrorCode(ex1);
            AddToLog("Read failure (Win32 error: {0}) at
{1:###,###,###,###,##0}-{2:###,###,###,###,##0}", errorCode1, sectorIndex,
sectorIndex + sectorCount - 1);
            if (errorCode1 != (int)Win32Error.ERROR_IO_DEVICE && errorCode1
!= (int)Win32Error.ERROR_CRC)
            {
                ioErrorOccured = true;
                return null;
            }
            byte[] data = new byte[sectorCount * m_disk.BytesPerSector];
            // Try to read sector by sector (to maximize data recovery)
            for (long sectorOffset = 0; sectorOffset < sectorCount;
sectorOffset++)
            {
                long currentPosition = (sectorIndex + sectorOffset) *
m_disk.BytesPerSector;
                UpdateStatus(currentPosition);
                try
                {
                    byte[] sectorBytes = m_disk.ReadSector(sectorIndex +
sectorOffset);
                    Array.Copy(sectorBytes, 0, data, sectorOffset *
m_disk.BytesPerSector, m_disk.BytesPerSector);
                }
                catch (IOException ex2)
                {
                    int errorCode2 =
IOExceptionHelper.GetWin32ErrorCode(ex2);
                    AddToLog("Read failure (Win32 error: {0}) at sector
{1:###,###,###,###,##0}", errorCode2, sectorIndex + sectorOffset);
                    if (errorCode2 == (int)Win32Error.ERROR_IO_DEVICE ||
errorCode2 == (int)Win32Error.ERROR_CRC)
                    {
                        damagedSectors.Add(sectorIndex + sectorOffset);
                    }
                    else // ERROR_FILE_NOT_FOUND, ERROR_DEVICE_NOT_CONNECTED,
ERROR_DEV_NOT_EXIST etc.
                    {
                        ioErrorOccured = true;
                        return null;
                    }
                }
                if (m_abort)
                {
                    return null;
                }
            }
            if (damagedSectors.Count == 0)
            {

```



```

namespace HardDiskValidator
{
    public class DiskTester : DiskReader
    {
        const int TransferSizeLBA = 131072; // 64 MB (assuming 512-byte sectors)
        private TestName m_testName;
        public DiskTester(TestName testName, Disk disk) : base(disk)
        {
            m_testName = testName;
        }
        public BlockStatus PerformTest(long sectorIndex, long sectorCount)
        {
            if (m_testName == TestName.Read)
            {
                return PerformReadTest(sectorIndex, sectorCount);
            }
            else if (m_testName == TestName.ReadWipeDamagedRead)
            {
                return PerformReadWipeDamagedReadTest(sectorIndex, sectorCount);
            }
            else if (m_testName == TestName.ReadWriteVerifyRestore)
            {
                return PerformReadWriteVerifyRestoreTest(sectorIndex,
sectorCount);
            }
            else if (m_testName == TestName.WriteVerify)
            {
                return PerformWriteVerifyTest(sectorIndex, sectorCount);
            }
            else if (m_testName == TestName.Write)
            {
                return PerformWriteTest(sectorIndex, sectorCount);
            }
            else
            {
                return PerformVerifyTest(sectorIndex, sectorCount);
            }
        }
        private BlockStatus PerformReadTest(long sectorIndex, long sectorCount)
        {
            // The only reason to read every sector separately is to maximize data
recovery
            for (long sectorOffset = 0; sectorOffset < sectorCount; sectorOffset
+= PhysicalDisk.MaximumDirectTransferSizeLBA)
            {
                long leftToRead = sectorCount - sectorOffset;
                int sectorsToRead = (int)Math.Min(leftToRead,
PhysicalDisk.MaximumDirectTransferSizeLBA);
                bool ioErrorOccured;
                byte[] segment = ReadSectors(sectorIndex + sectorOffset,
sectorsToRead, out ioErrorOccured);
                if (Abort)
                {
                    return BlockStatus.Untested;
                }
                if (ioErrorOccured)
                {
                    return BlockStatus.IOError;
                }

                if (segment == null)
                {
                    return BlockStatus.Damaged;
                }
            }
        }
    }
}

```

```

        }
    }
    return BlockStatus.OK;
}

private BlockStatus PerformReadWipeDamagedReadTest(long sectorIndex, long
sectorCount)
{
    bool crcErrorOccuredInFirstPass = false;
    // We use a large TransferSizeLBA to circumvent the disk caching
mechanism
    for (long sectorOffset = 0; sectorOffset < sectorCount; sectorOffset
+= TransferSizeLBA)
    {
        long leftToRead = sectorCount - sectorOffset;
        int sectorsToRead = (int)Math.Min(leftToRead, TransferSizeLBA);
        List<long> damagedSectors;
        bool ioErrorOccured;
        // Clear allocations from previous iteration
        GC.Collect();
        GC.WaitForPendingFinalizers();
        ReadEverySector(sectorIndex + sectorOffset, sectorsToRead, out
damagedSectors, out ioErrorOccured);
        if (Abort)
        {
            return BlockStatus.Untested;
        }
        if (ioErrorOccured)
        {
            return BlockStatus.IOError;
        }
        if (damagedSectors.Count > 0)
        {
            crcErrorOccuredInFirstPass = true;
            foreach (long damagedSectorIndex in damagedSectors)
            {
                byte[] sectorBytes = new byte[Disk.BytesPerSector];
                try
                {
                    WriteSectors(damagedSectorIndex, sectorBytes);
                }
                catch (IOException ex)
                {
                    int errorCode =
System.Runtime.InteropServices.Marshal.GetHRForException(ex);
                    AddToLog("Write failure (Win32 error: {0}) at sector
{1:###,###,###,###,##0}", errorCode, damagedSectorIndex);
                    return BlockStatus.IOError;
                }
                AddToLog("Sector {0:###,###,###,###,##0} has been
overwritten", damagedSectorIndex);
            }
            // TODO: We should make sure that the writes are flushed to
disk

            byte[] segment = ReadSectors(sectorIndex + sectorOffset,
sectorsToRead, out ioErrorOccured);

            if (Abort)
            {
                return BlockStatus.Untested;
            }
            if (ioErrorOccured)
            {
                return BlockStatus.IOError;
            }
        }
    }
}

```

```

        }
        if (segment == null)
        {
            return BlockStatus.Damaged;
        }
    }
    if (Abort)
    {
        return BlockStatus.Untested;
    }
}
if (!crcErrorOccuredInFirstPass)
{
    return BlockStatus.OK;
}
else
{
    return BlockStatus.OverwriteOK;
}
}
private BlockStatus PerformReadWriteVerifyRestoreTest(long sectorIndex,
long sectorCount)
{
    bool crcErrorOccuredInFirstPass = false;
    for (long sectorOffset = 0; sectorOffset < sectorCount; sectorOffset
+= TransferSizeLBA)
    {
        long leftToRead = sectorCount - sectorOffset;
        int sectorsToRead = (int)Math.Min(leftToRead, TransferSizeLBA);
        List<long> damagedSectors;
        bool ioErrorOccured;
        // Clear allocations from previous iteration
        GC.Collect();
        GC.WaitForPendingFinalizers();
        byte[] data = ReadEverySector(sectorIndex + sectorOffset,
sectorsToRead, out damagedSectors, out ioErrorOccured);
        if (Abort)
        {
            return BlockStatus.Untested;
        }
        if (ioErrorOccured)
        {
            return BlockStatus.IOError;
        }
        if (damagedSectors.Count > 0)
        {
            crcErrorOccuredInFirstPass = true;
        }
        // TODO: generate dummy read operation if leftToRead <
TransferSizeLBA
        BlockStatus writeVerifyStatus =
PerformWriteVerifyTest(sectorIndex + sectorOffset, sectorsToRead);

        // restore the original data
        try
        {
            WriteSectors(sectorIndex + sectorOffset, data);
        }
        catch (Exception ex)
        {
            int errorCode =
System.Runtime.InteropServices.Marshal.GetHRForException(ex);

```

```

        AddToLog("Restore failure (Win32 error: {0}) at sectors
{1:###,###,###,###,##0}-{2:###,###,###,###,##0}",      errorCode,      sectorIndex,
sectorIndex + sectorsToRead - 1);
        return BlockStatus.IOError;
    }
    if (writeVerifyStatus != BlockStatus.OK)
    {
        return writeVerifyStatus;
    }
    if (Abort)
    {
        return BlockStatus.Untested;
    }
}
if (crcErrorOccuredInFirstPass)
{
    return BlockStatus.OverwriteOK;
}
else
{
    return BlockStatus.OK;
}
}
private BlockStatus PerformWriteVerifyTest(long sectorIndex, long
sectorCount)
{
    // When we perform a write operation and then immediately read it, we
    may get the result from the disk buffer and not from the disk surface (confirmed
    with a broken WD15EADS)
    // To be sure that we read from the disk surface, we first write the
    entire UI block, and only then read from it.
    for (long sectorOffset = 0; sectorOffset < sectorCount; sectorOffset
+= PhysicalDisk.MaximumDirectTransferSizeLBA)
    {
        long leftToRead = sectorCount - sectorOffset;
        int sectorsToRead = (int)Math.Min(leftToRead,
PhysicalDisk.MaximumDirectTransferSizeLBA);
        // Clear allocations from previous iteration
        GC.Collect();
        GC.WaitForPendingFinalizers();
        byte[] pattern = GetTestPattern(sectorIndex + sectorOffset,
sectorsToRead, Disk.BytesPerSector);
        try
        {
            WriteSectors(sectorIndex + sectorOffset, pattern);
        }
        catch (IOException ex)
        {
            int errorCode =
System.Runtime.InteropServices.Marshal.GetHRForException(ex);
            AddToLog("Write failure (Win32 error: {0}) at sectors
{1:###,###,###,###,##0}-{2:###,###,###,###,##0}",      errorCode,      sectorIndex,
sectorIndex + sectorsToRead - 1);
            return BlockStatus.IOError;
        }
        if (Abort)
        {
            return BlockStatus.Untested;
        }
    }
    return PerformVerifyTest(sectorIndex, sectorCount);
}
private BlockStatus PerformWriteTest(long sectorIndex, long sectorCount)

```

```

    {
        for (long sectorOffset = 0; sectorOffset < sectorCount; sectorOffset
+= PhysicalDisk.MaximumDirectTransferSizeLBA)
        {
            int leftToWrite = (int)(sectorCount - sectorOffset);
            int sectorsToWrite = (int)Math.Min(leftToWrite,
PhysicalDisk.MaximumDirectTransferSizeLBA);
            // Clear allocations from previous iteration
            GC.Collect();
            GC.WaitForPendingFinalizers();
            byte[] pattern = GetTestPattern(sectorIndex + sectorOffset,
sectorsToWrite, Disk.BytesPerSector);
            try
            {
                WriteSectors(sectorIndex + sectorOffset, pattern);
            }
            catch (IOException ex)
            {
                int errorCode =
System.Runtime.InteropServices.Marshal.GetHRForException(ex);
                AddToLog("Write failure (Win32 error: {0}) at sectors
{1:###,###,###,###,##0}-{2:###,###,###,###,##0}", errorCode, sectorIndex,
sectorIndex + sectorsToWrite - 1);
                return BlockStatus.IOError;
            }
            if (Abort)
            {
                return BlockStatus.Untested;
            }
        }
        return BlockStatus.OK;
    }
    private BlockStatus PerformVerifyTest(long sectorIndex, long sectorCount)
    {
        bool verificationMismatch = false;
        for (long sectorOffset = 0; sectorOffset < sectorCount; sectorOffset
+= PhysicalDisk.MaximumDirectTransferSizeLBA)
        {
            long leftToRead = sectorCount - sectorOffset;
            int sectorsToRead = (int)Math.Min(leftToRead,
PhysicalDisk.MaximumDirectTransferSizeLBA);
            // Clear allocations from previous iteration
            GC.Collect();
            GC.WaitForPendingFinalizers();
            bool ioErrorOccured;
            byte[] buffer = ReadSectors(sectorIndex + sectorOffset,
sectorsToRead, out ioErrorOccured);
            if (Abort)
            {
                return BlockStatus.Untested;
            }
            if (buffer == null)
            {
                return BlockStatus.Damaged;
            }
            for (int position = 0; position < sectorsToRead; position++)
            {
                byte[] pattern = GetTestPattern(sectorIndex + sectorOffset +
position, Disk.BytesPerSector);
                byte[] sectorBytes = ByteReader.ReadBytes(buffer, position *
Disk.BytesPerSector, Disk.BytesPerSector);
                if (!ByteUtils.AreByteArraysEqual(pattern, sectorBytes))
                {

```

```

        verificationMismatch = true;
        AddToLog("Verification mismatch at sector
{0:###,###,###,###,##0}", sectorIndex + sectorOffset + position);
    }
}
}
if (verificationMismatch)
{
    return BlockStatus.Damaged;
}
public void WriteSectors(long sectorIndex, byte[] data)
{
    int sectorCount = data.Length / Disk.BytesPerSector;
    if (sectorCount > PhysicalDisk.MaximumDirectTransferSizeLBA)
    {
        // we must write one segment at the time
        for (int sectorOffset = 0; sectorOffset < sectorCount;
sectorOffset += PhysicalDisk.MaximumDirectTransferSizeLBA)
        {
            int leftToWrite = sectorCount - sectorOffset;
            int sectorsToWrite = (int)Math.Min(leftToWrite,
PhysicalDisk.MaximumDirectTransferSizeLBA);
            byte[] segment = new byte[sectorsToWrite *
Disk.BytesPerSector];
            Array.Copy(data, sectorOffset * Disk.BytesPerSector, segment,
0, sectorsToWrite * Disk.BytesPerSector);
            long currentPosition = (sectorIndex + sectorOffset) *
Disk.BytesPerSector;
            UpdateStatus(currentPosition);
            Disk.WriteSectors(sectorIndex + sectorOffset, segment);
        }
    }
    else
    {
        UpdateStatus(sectorIndex * Disk.BytesPerSector);
        Disk.WriteSectors(sectorIndex, data);
    }
}
private static byte[] GetTestPattern(long sectorIndex, int
bytesPerSector)
{
    return GetTestPattern(sectorIndex, 1, bytesPerSector);
}
private static byte[] GetTestPattern(long sectorIndex, int sectorCount,
int bytesPerSector)
{
    byte[] buffer = new byte[sectorCount * bytesPerSector];
    for (int sectorOffset = 0; sectorOffset < sectorCount; sectorOffset++)
    {
        byte[] pattern = BigEndianConverter.GetBytes(sectorIndex +
sectorOffset);
        for (int offsetInSector = 0; offsetInSector <= bytesPerSector -
8; offsetInSector += 8)
        {
            Array.Copy(pattern, 0, buffer, sectorOffset * bytesPerSector
+ offsetInSector, 8);
        }
    }
    return buffer;
}
}
}

```

ДОДАТОК Б
Копії слайдів презентації

ДИПЛОМНА РОБОТА

РОЗРОБКА ПРОГРАМИ ДЛЯ МОНІТОРИНГУ СТАНУ ЖОРСТКОГО ДИСКУ

Розробив: ст. групи КНТ-218сп
Керівник: доцент, к.т.н.

Тарасенко Б.В.
Сердюк С.М.

Рисунок Б.1 – Слайд №1

Мета роботи

розробка програми, яка дозволить переглядати значення S.M.A.R.T. атрибутів та проводити перевірку на наявність пошкоджених секторів обраного жорсткого диску (ЖД)

Об'єкт дослідження – процес перевірки стану ЖД

Предмет дослідження – методи перевірки стану ЖД з використанням технології S.M.A.R.T.



НУ “ЗП”, Кафедра програмних засобів

Рисунок Б.2 – Слайд №2

Задачі, що вирішуються

- огляд технології S.M.A.R.T.;
- огляд існуючих програмних аналогів;
- розробка технічного завдання;
- розробка діаграми класів програми;
- розробка програмних модулів застосунку;
- розробка інтерфейсу користувача;
- розробка необхідної документації.



НУ “ЗП”, Кафедра програмних засобів

3

Рисунок Б.3 – Слайд №3

Перелік S.M.A.R.T. атрибутів, що ідентифікуються програмою

Номер	Назва атрибуту
1	Помилки зчитування даних з диска
3	Час розкручування пакету дисків (мс)
4	Кількість запуску/зупинки шпинделя
7	Помилки позиціонування блоку магнітних головок
9	Час роботи (год)
10	Повторні спроби запуску ЖД
12	Кількість включень ЖД
13	Кількість помилок при зчитуванні
188	Кількість перерваних операцій через HDD тайм-аут
189	Кількість випадків запису при висоті головки вище стандартної
191	Кількість помилок через ударні навантаження
192	Відмова відключення живлення
193	Кількість циклів завантаження/розвантаження блоку головок
194	Температура для механічної частини диска
198	Невиправні помилки секторів
200	Кількість помилок під час запису
207	Сила струму при розкручуванні диску
220	Дистанція зміщення блоку дисків щодо шпинделя (струс диску)
223	Повтори вивантажень/завантажень блоку магнітних головок
226	Час завантаження головок (мс)

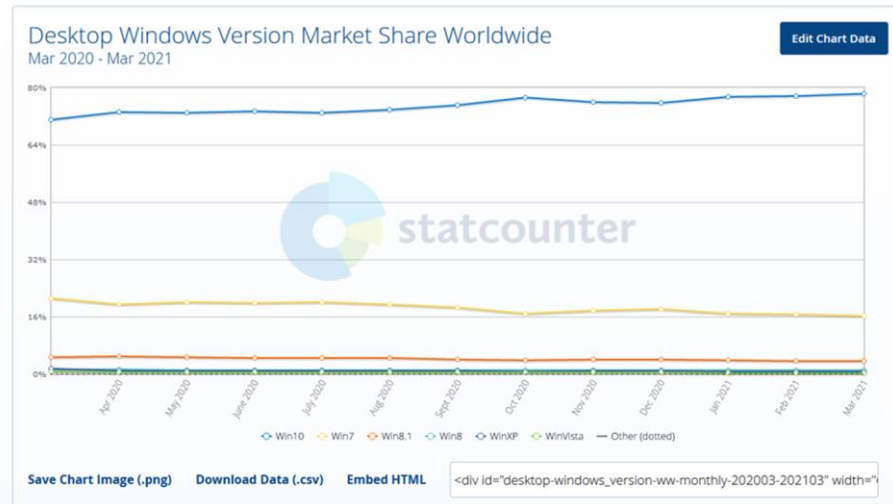


НУ “ЗП”, Кафедра програмних засобів

4

Рисунок Б.4 – Слайд №4

Світова частка ринку настільних версій ОС Windows

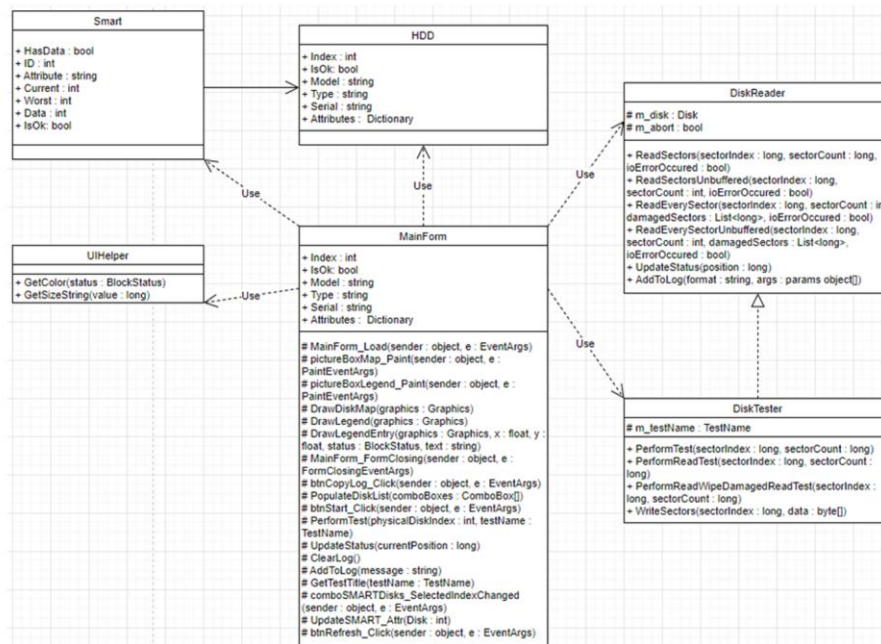


НУ "ЗП", Кафедра програмних засобів

Рисунок Б.5 – Слайд №5

5

Діаграма класів програми



НУ "ЗП", Кафедра програмних засобів

Рисунок Б.6 – Слайд №6

6

Код та опис методів класу DiskReader

```
public class DiskReader
// функція для зчитування секторів
    ReadSectors(long sectorIndex, long sectorCount, out bool ioErrorOccurred)
// допоміжна функція для зчитування секторів
    ReadSectorsUnbuffered(long sectorIndex, int sectorCount, out bool
ioErrorOccurred)
// функція для зчитування певного сектору
    ReadEverySector(long sectorIndex, int sectorCount, out List<long>
damagedSectors, out bool ioErrorOccurred)
// допоміжна функція для зчитування певного сектору
    ReadEverySectorUnbuffered(long sectorIndex, int sectorCount, out
List<long> damagedSectors, out bool ioErrorOccurred)
// функція для оновлення статусу блоку на карті ЖД
    UpdateStatus(long position)
// функція для додання інформації до логу
    AddToLog(string format, params object[] args)
```

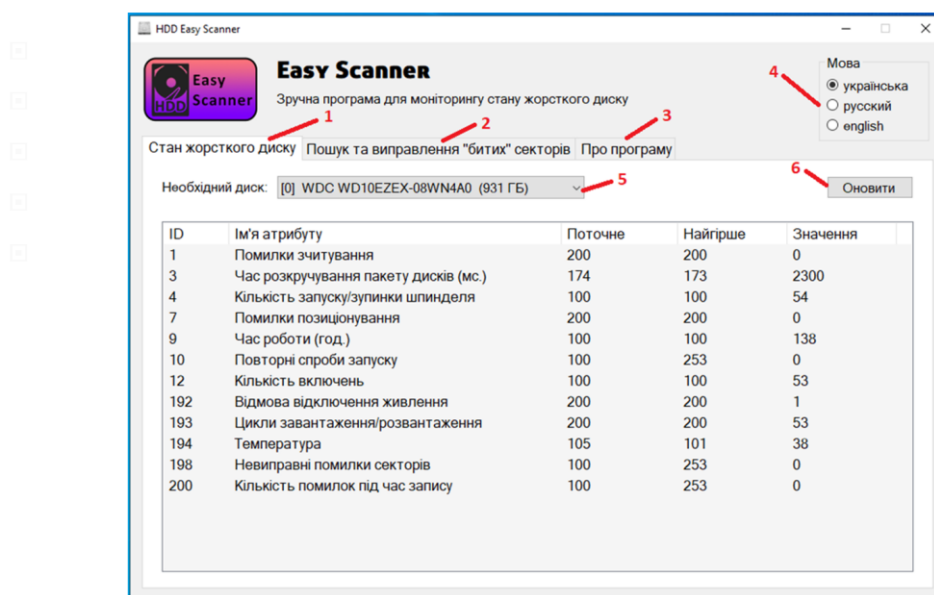


НУ "ЗП", Кафедра програмних засобів

7

Рисунок Б.7 – Слайд №7

Головна форма з позначками кнопок керування

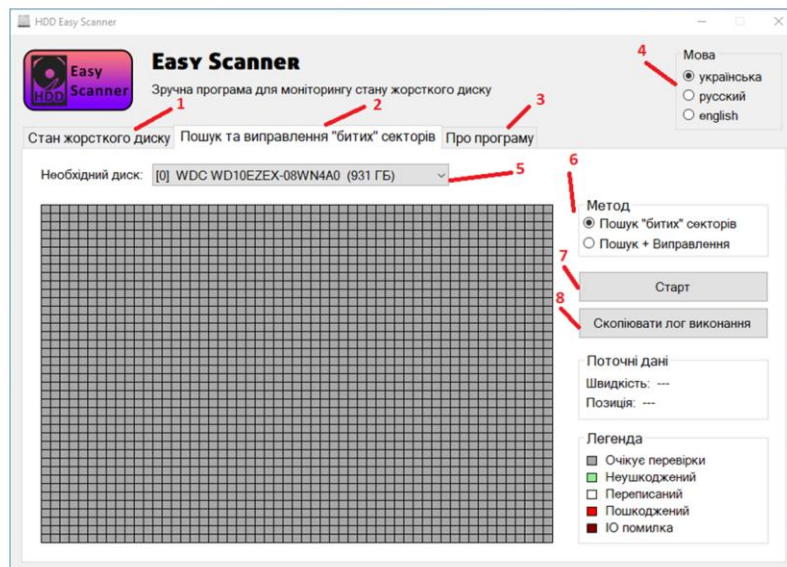


НУ "ЗП", Кафедра програмних засобів

8

Рисунок Б.8 – Слайд №8

Вкладка «Пошук та виправлення "битих" секторів» з позначками кнопок керування

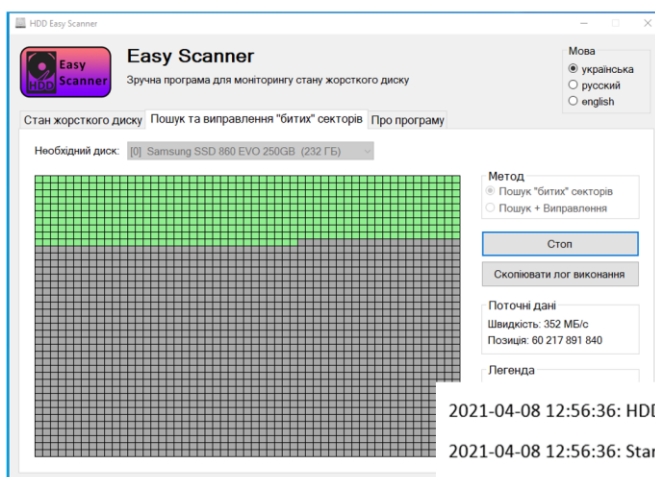


НУ "ЗП", Кафедра програмних засобів

Рисунок Б.9 – Слайд №9

9

Результати тесту «Пошук "битих" секторів»



2021-04-08 12:56:36: HDD Easy Scanner

2021-04-08 12:56:36: Starting Read Test

2021-04-08 12:56:36: Disk: Samsung SSD 860 EVO 250GB, S/N: S4G2NJ0N226459L

2021-04-08 12:56:36: Disk size: 232 ГБ (488 397 168 sectors, 512 bytes per sector)

2021-04-08 13:07:46: Test Completed



НУ "ЗП", Кафедра програмних засобів

Рисунок Б.10 – Слайд №10

10

Інтерфейс вкладки «Про програму»

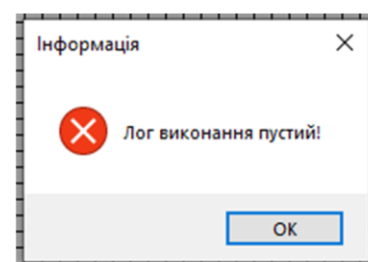
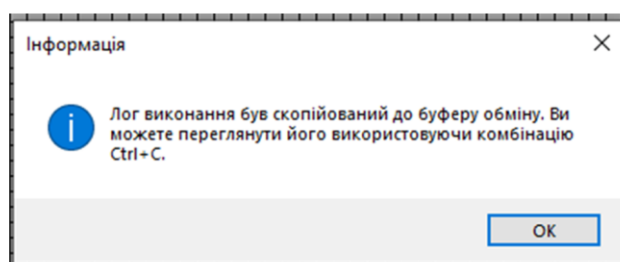


НУ "ЗП", Кафедра програмних засобів

11

Рисунок Б.11 – Слайд №11

Повідомлення оператору



НУ "ЗП", Кафедра програмних засобів

12

Рисунок Б.12 – Слайд №12

Висновки

- проведено огляд технології S.M.A.R.T.;
- проведено огляд існуючих програмних аналогів;
- розроблено технічне завдання;
- розроблено діаграму класів;
- розроблено програмні модулі застосунку;
- розроблено інтерфейс користувача;
- розроблено необхідну документацію.



НУ “ЗП”, Кафедра програмних засобів

Рисунок Б.13 – Слайд №13

13

**Доклад закінчено,
дякую за увагу!**



НУ “ЗП”, Кафедра програмних засобів

Рисунок Б.14 – Слайд №14

14