Центральноукраїнський національний технічний університет
Механіко-технологічний факультет
Кафедра кібербезпеки та програмного забезпечення

”Допущено до захисту”
Завідувач кафедри кібербезпеки
та програмного забезпечення
д.т.н., професор
_____ Олексій СМІРНОВ
“____” _____ 2026 р.

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
за першим (бакалаврським) рівнем вищої освіти
на тему
“Програмне забезпечення системи інтелектуального підвищення відказостійкості периферійної ЦОД”

Виконав здобувач вищої освіти
IV курсу, групи КН-22
ОПП «Комп’ютерні науки»
спеціальності 122 «Комп’ютерні науки»
_____ Калашник О.В.
«____» _____ 2026 р.

Керівник проекту
кандидат технічних наук
_____ Лисенко І.А.
«____» _____ 2026 р.
Рецензент _____
_____

м. Кропивницький

Центральноукраїнський національний технічний університет

Факультет *Механіко-технологічний*

Кафедра *Кібербезпеки та програмного забезпечення*

Освітній ступінь *бакалавр*

Галузь знань ._12 _____*“Інформаційні технології”*_____

Спеціальність 122 *“Комп’ютерні науки”*

Освітньо-професійна (освітньо-наукова) програма _*“Комп’ютерні науки”*_

ЗАТВЕРДЖУЮ
Завідувач кафедри
д.т.н., проф.
_____ Олексій СМІРНОВ
« 06 » лютого 2026 року

ЗАВДАННЯ НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ РОБОТУ ЗА ПЕРШИМ (БАКАЛАВРСЬКИМ) РІВНЕМ ВИЩОЇ ОСВІТИ ЗДОБУВАЧА ВИЩОЇ ОСВІТИ

Калашнику Олександру Володимировичу
(прізвище, ім’я, по батькові)

1. Тема роботи *Програмне забезпечення системи інтелектуального підвищення відказостійкості периферійної ЦОД*

2. Керівник роботи *Лисенко Ірина Анатоліївна, канд. техн. наук*
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу № 116-02 від 06.02.2026 року

3. Строк подання студентом роботи до захисту *23.05.2026 р.*

4. Мета та завдання випускної кваліфікаційної роботи*: Метою роботи є розробка програмного забезпечення системи інтелектуального підвищення відказостійкості периферійної ЦОД*

5. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Призначення та область використання.

2.Перегляд аналогічних існуючих систем.

3. Опис і обґрунтування проектних рішень.

4. Етапи програмування системи.

5. Впровадження системи в промислову експлуатацію.

6. Висновки

6. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

Структурна схема системи	*1 аркуш*
Функціональна схема системи	*1 аркуш*
Діаграма процесів	*1 аркуш*
Блок-схема алгоритму роботи додатку	*2 аркуша*

7. Дата видачі завдання « 06 » лютого 2026 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти	Строк виконання етапів випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти	Примітка
1.	Аналіз існуючих систем	10.03.2026 р.	
2.	Постановка задачі, оформлення ТЗ	15.03.2026 р.	
3.	Розробка моделі компонента	20.03.2026 р.	
4.	Розробка структур даних	25.03.2026 р.	
5.	Розробка алгоритмів зв’язку та відображення	30.03.2026 р.	
6.	Програмування алгоритмів	10.04.2026 р.	
7.	Оформлення ПЗ	17.04.2026 р.	
8.	Попередній захист роботи	23.05.2026 р.	

Дата видачі завдання
« 06 » лютого 2026 р.

Підпис керівника
_____ Лисенко І.А.
(прізвище та ініціали)

Завдання прийнято до виконання
« 06 » лютого 2026 р.

Підпис здобувача
_____ Калашник О.В.
(прізвище та ініціали)

АНОТАЦІЯ

Калашник О.В. Програмне забезпечення системи інтелектуального підвищення відказостійкості периферійної ЦОД. 122 Комп'ютерні науки. Центральноукраїнський національний технічний університет. Кропивницький. 2026.

В даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи інтелектуального підвищення відказостійкості периферійної ЦОД.

Метою розробки є програмне забезпечення системи інтелектуального підвищення відказостійкості периферійної ЦОД.

Результат роботи – програмна реалізація системи інтелектуального підвищення відказостійкості периферійної ЦОД.

В процесі роботи над програмною моделлю виконано аналіз існуючих апаратних та програмних засобів. В повній мірі описані всі компоненти розробленого програмного забезпечення.

Розроблено зручний інтерфейс користувача. Наведені інструкції по роботі з програмними засобами.

Програма може використовуватися на ПЕОМ з ОС Windows 10/11.

Програму розроблено в середовищі Visual C#.

Ключові слова: комп'ютерні науки, інтелектуальне підвищення відказостійкості периферійної ЦОД

ABSTRACT

Kalashnyk O.V. Software for the system of intellectual increase of fault tolerance of peripheral data center. 122 Computer Science. Central Ukrainian National Technical University. Kropyvnytskyi. 2026.

In this final qualification work for the first (bachelor's) level of higher education, software has been developed, which is intended for the system of intellectual increase of fault tolerance of peripheral data center.

The purpose of the development is software for the system of intellectual increase of fault tolerance of peripheral data center.

The result of the work is the software implementation of the system of intellectual increase of fault tolerance of peripheral data center.

In the process of working on the software model, an analysis of existing hardware and software was performed. All components of the developed software are fully described.

A convenient user interface has been developed. Instructions for working with software are provided.

The program can be used on a PC with Windows 10/11 OS.

The program was developed in the Visual C# environment.

Keywords: computer science, intelligent enhancement of edge data center resiliency

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ 2
ВСТУП..... 3
1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ 5
1.1 Призначення системи..... 5
1.2 Область застосування..... 5
2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ 9
2.1 Огляд існуючих систем, технологій, архітектур та програмних рішень за профілем теми випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти..... 9
2.2 Обґрунтування вибору засобів для побудови системи та мови програмування..... 17
2.3 Розгорнута постановка завдання 20
3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ 22
3.1 Опис функціонування системи 22
3.2 Розробка структурної схеми..... 29
3.3 Розробка функціональної схеми 35
3.4 Розробка діаграми процесів..... 39
4 РЕАЛІЗАЦІЯ РОБОТИ. РОЗРАХУНКИ І ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ВІРНІСТЬ ПРОЕКТНИХ ТА ПРОГРАМНИХ РІШЕНЬ..... 41
4.1 Розробка блок-схем та опис алгоритмів функціонування системи..... 41
4.2 Захист розробленого програмного забезпечення..... 57
5 ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ..... 58
6 ОСНОВНІ ВИСНОВКИ..... 62
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 64

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

АБГШ	–	аддітивний білий гаусів шум
ЕОМ	–	електронно-обчислювальна машина
ІМС	–	інтегральна мікросхема
ARQ	–	автоматичний запит повторної передачі
CD-DA	–	Compact Disc Digital Audio
CIRC	–	Cross Interleaved Reed Solomon Code
EAB	–	Embedded Array Block, блок зосередженої пам'яті
ECC	–	error-correcting code, код корекції помилок
FEC	–	метод прямої корекції помилок
LDPC	–	коди Галлагера
NAK	–	негативне підтвердження

ВСТУП

Актуальність теми. Зменшення кількості покладених на локальний ЦОД функцій і зниження потужності зовсім не означають, що його роль стає другорядною. На жаль, сьогодні в дизайні більшості периферійних ЦОД є вади, що приводить до дорогих простоїв.

Від колишнього центра обробки даних потужністю 1 Мвт, що раніше перебував на території філії компанії, зараз може залишитися пара стійок з IT-устаткуванням, що забезпечує виконання найбільш важливих застосунків і зв'язок із хмарою. Зменшення кількості покладених на локальний ЦОД функцій і зниження потужності зовсім не означають, що його роль стає другорядною. Навпаки, найчастіше те, що компанії залишають у себе, і є найважливішим. На жаль, сьогодні в дизайні більшості периферійних ЦОД є вади, що приводить до дорогих простоїв. Для досягнення максимальної віддачі від інвестицій необхідний систематичний підхід до оцінки рівня доступності ЦОД у гібридному середовищі, для того щоб забезпечити найбільшу економічну ефективність інвестицій.

Розвиток Інтернету речей, що росте обсяг трафіку й розширення області застосування хмарних рішень є основними технологічними тенденціями, що міняють подання про ЦОД. У великих і хмарних центрах обробки даних сьогодні розміщається безліч важливих для бізнесу застосунків. Однак не всі вони переносяться в хмару, і причин тому безліч: державні галузеві норми регулювання, корпоративна політика, використання пропрієтарних застосунків, затримки в процесі передачі даних і багато чого іншого.

У результаті формується так зване гібридне середовище ЦОД, що вибудовується в наступну ієрархію: централізовані хмарні ЦОД, середні й великі регіональні ЦОД, локальні малі ЦОД. У даній роботі описана найпоширеніша практика використання трьох типів ЦОД, обговорюється зміна очікувань щодо

доступності встаткування, запропонований метод оцінки необхідного рівня відказостійкості для локальних ЦОД, що сприяють досягненню цілей бізнесу, а також перераховані кращі приклади застосування мікроцентрів обробки даних на периферії.

Мета й завдання дослідження. Метою роботи є програмне забезпечення системи інтелектуального підвищення відказостійкості периферійної ЦОД.

Для досягнення поставленої мети визначена програма дослідження, що складається з наступних завдань:

– Огляд існуючих систем інтелектуального підвищення відказостійкості периферійної ЦОД.

– Дослідження системи інтелектуального підвищення відказостійкості периферійної ЦОД.

– Програмна реалізація системи інтелектуального підвищення відказостійкості периферійної ЦОД.

Практична цінність отриманих результатів полягає в тому, що розроблені алгоритми дозволяють успішно вирішувати задачі інтелектуального підвищення відказостійкості периферійної ЦОД.

Таким чином, виходячи з вищеперерахованого, програмне забезпечення системи інтелектуального підвищення відказостійкості периферійної ЦОД, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти.

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1 Призначення системи

Централізована хмарна система створювалася розраховуючи на застосунки, для яких затримка не мала критичного значення (електронна пошта, онлайн-платежі, соціальні мережі). Однак з переносом важливих застосунків у хмару стало очевидно, що проблеми затримки, обмеження пропускної здатності, дотримання безпеки й нормативних вимог вимагають рішення. Наприклад, при експлуатації безпілотних автомобілів будь-які відхилення від заданих параметрів можуть привести до аварії. Очевидною стала необхідність переносу обчислень ближче до джерела даних і місцю їхнього використання.

Розподіл контента – ще одна область застосування, де завдяки обробці даних у безпосередній близькості від споживачів скорочуються витрати на канальну ємність і підвищується якість потокового віщання.

Деяким підприємствам найчастіше потрібно зберегти в себе важливі для бізнесу застосунка. Це дозволяє краще контролювати їхню роботу, дотримувати державних галузевих норм регулювання й забезпечувати доступність. Іноді такі застосунки дублюються в хмарі з метою резервування.

1.2 Область застосування

Централізований ЦОД

Великі централізовані ЦОД потужністю кілька мегаватів нерідко створюються як платформа для рішення критично важливих завдань, тому особлива увага звертається на доступність. Для запобігання аварій застосовується ряд визнаних практик, використовуваних уже багато років. Всі дії обслуговуючого персоналу спрямовані на забезпечення безперервної й

ефективної роботи всіх систем 24 години на добу й 7 днів у тиждень. Крім того, такі ЦОД часто проектуються й іноді сертифікуються у відповідності зі стандартами Uptime Institute Tier 3 або Tier 4.

Розповсюджені кращі практики містять у собі:

– Резервування критичних систем. Найбільш важливі системи живлення й охолодження проектуються з надмірністю (часто 2N) щоб уникнути простоїв при несправності або проведенні технічного обслуговування.

– Високий рівень фізичної безпеки. Широке поширення одержали біометричні датчики на дверях, кабіни КПП, відеоспостереження й цілодобова охорона, які забезпечують безпека систем і доступ тільки авторизованого персоналу.

– Рядне розташування стійок. Крім обмеження фізичного доступу до шаф для зниження ймовірності помилки з вини людини (від'єднання не того проведення, підключення обох блоків живлення до одного променя й т.п.), упорядковуються силові й мережні кабелі. Розподіл повітряних потоків організується таким чином, щоб виключити виникнення крапок перегріву: у невикористовувані посадкові місця встановлюються панелі-заглушки, у технологічні отвори – щіткові ущільнювачі й т.д.

– Моніторинг. Для того щоб системи керування інфраструктурою ЦОД (DCIM) і будинком (BMS) могли здійснювати контроль і оптимізацію всіх систем ЦОД, установлюються датчики й вимірювальні прилади.

Регіональний ЦОД

Регіональні ЦОД менше по розмірі, чим великі централізовані центри обробки даних, і розміщаються там, де інформація генерується й використовується. Як згадувалося вище, ці ЦОД призначені для застосунків, чутливих до затримки передачі даних або вимогливих до пропускної здатності. Їхнє стратегічне розташування вибирається таким чином, щоб оптимізувати обробку більших обсягів даних. Такі площадки можна зрівняти з «мостом» між центральними й локальними ЦОД на місцях.

Як і великі централізовані ЦОД, регіональні звичайно спроектовані з урахуванням забезпечення безпеки й доступності даних. Дизайн подібних об'єктів часто відповідає стандартам Uptime Institute Tier 3. Іноді при їхньому створенні застосовуються рішення високої заводської готовності, а як відправна крапка можуть використовуватися референсні варіанти дизайну.

Локальний ЦОД

Локальний ЦОД – це центр збору й обробки даних, розташований у тім же місці, де перебувають його користувачі. Для опису цього типу ЦОД застосовуються різні терміни – наприклад, «внутрикорпоративний центр обробки даних» або «мікроЦОД». Локальні ЦОД можуть варіюватися по потужності: від 1-2 Мвт до всього лише 10-20 кВт. Оскільки число бізнес-застосунків, переносимих у хмару або на площадки комерційних ЦОД, постійно росте, розмір локального ЦОД зменшується: іноді він може складатися з пари стійок, розташованих у невеликій кімнаті або в корпусі.

Багато які з таких сучасних невеликих ЦОД проектуються з дотриманням лише мінімальних вимог до резервування й доступності, тобто відповідають рівню Tier 1.

Недоліками таких локальних ЦОД є:

– Низький рівень безпеки: у приміщення можливий доступ сторонніх осіб, стійки відкриті (не мають дверей).

– Неупорядковані стійки: про упорядкувані проводів замислюються в останню чергу, що приводить до їхнього заплутування, проблемам з вентиляцією усередині стійок, збільшенню числа людських помилок під час додавання, переміщення, зміни компонентів.

– Відсутність резервування: системи живлення (ІБП, система розподілу живлення) часто мають конфігурацію N, що знижує їхня доступність і здатність підтримувати працездатність центра під час технічного обслуговування.

– Відсутність спеціальної системи охолодження: у малих приміщеннях і шафах нерідко використовується загальна система пасивної вентиляції, через що можливий перегрів устаткування.

– Відсутність DCIM: приміщення найчастіше залишаються без догляду обслуговуючого персоналу або не контролюються за допомогою програмного забезпечення, що управляло б устаткуванням і запобігало збоям у роботі систем.

Таким чином, виходячи з вищеперерахованого, програмне забезпечення системи інтелектуального підвищення відказостійкості периферійної ЦОД, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти.

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

2.1 Огляд існуючих систем, технологій, архітектур, програмних рішень за профілем теми випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти

Цифрова економіка в її нинішньому виді далеко не досконала. Як ні іронічно, багато процесів і процедури обслуговування інфраструктури ЦОД, використовуваної для забезпечення працездатності цієї самої цифрової економіки, усе ще не оцифровані. Це виливається крім іншого й в аварії ЦОД, які, у свою чергу, обертаються серйозним репутаційним і фінансовим збитком для власників таких серверних ферм і їхніх корпоративних клієнтів.

Щоб підтвердити справедливість цього твердження наведемо як приклад інформацію про свіжі інциденти в ЦОД таких компаній як Vocus і NHS.

Халтурне технічне обслуговування системи ІБП викликало збої в центрі обробки даних Vocus

Погано виконана робота з обслуговування системи ІБП комерційного центра обробки даних в австралійському Сіднеї, штат Новий Південний Уельс, привела до перебоїв у його роботі. Аварія відбулася в одному з місцевих ЦОД компанії Vocus Communications.

Вона траплялася 13 лютого 2018 року в ранні годинники ранку. Через інцидент один з великих клієнтів цього колокейшн-провайдеру в особі компанії Servers Australia ухвалив рішення щодо переміщенні всього свого встаткування з даної північної ферми в іншу протягом наступних трьох місяців.

Аварія відбулася в ЦОД, що перебуває в сиднейском пригороді Олександрія. Перебої в роботі силової інфраструктури цього дата-центра почалися приблизно в 7 ранку за місцевим часом (AEST). Усунути неполадки вдалося в той же день.

Інцидент відбувся в складний період для колокейшн-провайдеру Vocus Communications, що недавно оголосив про намір продати 20 австралійських дата-центрів і весь свій бізнес у Новій Зеландії. Очікується, що продажу компенсують збитки в розмірі близько 100 млн. австралійських доларів (78,7 млн. доларів США) за підсумками минулого року, які в основному пояснюються невдалою інтеграцією недавно придбаних компаній Nextgen Networks, Amcom і M2.

Валлійські лікарі не змогли одержати доступ до електронних карток пацієнтів після відключення дата-центрів NHS

Схожий інцидент відбувся 25 січня 2018 року в З'єднаному Королівстві Великобританії й Північної Ірландії. Через технічні проблеми в роботі інфраструктури дата-центрів компанії NHS Wales лікарі загальної практики по всьому Уельсу виявилися не мають змоги одержувати доступ до документів про пацієнтів.

Неполадки були зафіксовані в роботі двох ЦОД. Обоє дата-центра, використовувані системою охорони здоров'я країни, були недоступні протягом 2 годин. Ці ЦОД розташовуються в Блаєнавоне й Кардіффе. Приблизно в 3 години дня за місцевим часом (GMT) команда NHS Wales повідомила, що «технічні проблеми» зачіпають два об'єкти, пообіцявши якнайшвидше їхнє усунення, назвавши це завдання «пріоритетним». Доступ до сервісів знову з'явився через дві години після аварії.

Прогнози по розвитку ЦОД

Життєвий цикл центрів обробки даних виміряється десятиліттями, але технології, використовувані в них, міняються постійно.

Сьогоднішні ЦОДи не відповідають устаткуванню, що перебуває в них усередині. Це нагадує розміщення сучасних компонентів iMac у корпусі комп'ютера Macintosh зразка 1984 року. Вони не занадто сполучаються один з одним. Вимоги до електроживлення, охолодженню й організації внутрішнього простору істотно змінилися.

Невідповідність це змушує організації проводити капітальну модернізацію своїх ЦОДів, приводити їх у порядок, щоб забезпечити необхідне прискорення виконання бізнес-операцій. По оцінках IDC, до 2027 року попит на прикладні системи наступного покоління й нові IT-архітектури змусить 55% організацій модернізувати наявні в них потужності або розгорнути нові ресурси.

Необхідність модернізації – один з десяти ключових прогнозів IDC відносно глобального ринку ЦОДів на найближчі три роки. В аналітичній компанії вважають, що галузь продовжить рух убік програмно-конфігуруємої інфраструктури, буде впроваджувати IT-інфраструктуру з автономним керуванням і розширювати використання моделей зі зміною оплати залежно від фактичного споживання.

У ЦОДи прийдуть технології й операційні моделі, адаптовані до більше швидких темпів змін. Вони почнуть впливати на фізичні об'єкти й самі ЦОДи. IT-організації усвідомлюють необхідність інновацій і надання ресурсів, що сприяють подальшому розвитку бізнесу, необхідність проведення перетворень зі швидкістю бізнесу, а не зі швидкістю, властивій традиційним IT. Все це зробить безпосередній вплив на рішення, пов'язані із ЦОДами, у тому числі на те, чиї ЦОДи має сенс використовувати, як проводити модернізацію і як платити за ресурси.

Перелічимо десять головних прогнозів IDC у відношенні ЦОДів на найближчу перспективу строком від одного до трьох років.

Модернізація ЦОДів

До 2027 року потреба в застосунках наступного покоління й нових IT-архітектурах у критично важливі для бізнесу областях змусить 55% підприємств модернізувати свої ЦОДи шляхом відновлення вже існуючих ресурсів і розгортання нових.

Раціоналізація робочого навантаження

До 2026 року 50% організацій займуться раціоналізацією робочого навантаження й прискоренням впровадження засобів нового покоління, що

приведе до корінних змін у проектуванні й розміщенні інфраструктури, а також у моделях IT-операцій.

Гібридні IT-операції.

До кінця 2026 року 70% компаній, залучених у цифрову трансформацію, для задоволення своїх бізнес-потреб стануть активніше проводити інвестиції в IT і становити операційні плани. Зміняться й підходи до наймання персоналу. Організація буде потрібно забезпечити наявність фахівців, що володіють необхідними для вибудовування цифрових бізнес-моделей і бізнес-процесів навичками.

Модульність

До 2027 року розширення сфери застосування енергоємних обчислювальних технологій змусить більшість операторів великих ЦОДів взяти на озброєння модульний підхід до розгортання ресурсів.

IT на вимогу

До 2027 року оплата ресурсів на основі їхнього фактичного споживання витиснуть традиційні підходи в міру вдосконалювання моделей надання ресурсів як сервіс. На них буде доводитися близько 40% від загального обсягу витрат підприємств на IT-інфраструктуру.

Тотальний контроль даних

До 2027 року 25% великих підприємств звернуть обов'язкові інвестиції у виконання нормативних вимог собі на користь завдяки повсюдному розгортанню засобів автоматизованого контролю даних у хмарах, ЦОДах і на периферійні (edge) системах.

Програмно-конфігуруємі IT

До кінця 2026 року потреба в підвищенні гнучкості, поліпшенні керованості й розширенні керування активами змусить компанії, залучені в процес цифрової трансформації, перевести понад половину IT-інфраструктури на програмно-конфігуруєму модель.

ЦОДи з інтелектуальними периферійними вузлами

До 2027 року в половині ЦОДів критично важлива інфраструктура буде автономною. Продовжиться й розгортання автономних IT на інтелектуальних периферійних вузлах. Організації будуть намагатися зв'язати ресурси ядра й периферійних вузлів мережі для підтримки ініціатив цифрової трансформації.

Цифрові кампуси

До 2027 року понад 50% компаній, що працюють зі споживчим ринком, будуть щорічно інвестувати у відновлення своїх мережних і обчислювальних ресурсів, а також ресурсів зберігання на периферійних вузлах мережі більше, ніж у модернізацію основних ЦОДів.

Гарантований рівень якості обслуговування

В 2026 році 60% цифрових сервісів не зможуть забезпечити бажаний рівень обслуговування клієнтів у силу своєї нездатності здійснювати ефективний моніторинг і швидко реагувати на зміну потреб у продуктивності й інших ресурсах в умовах зниження цін.

З розвитком хмарних систем планка швидкості й гнучкості піднята дуже високо. Внутрішні IT необхідно перетворювати до сервісного типу, а в рамках традиційних ЦОДів зробити це дуже складн».

Модернізація центрів обробки даних

Проекти модернізації дозволять компаніям перейти на більше прозорі операційні моделі й забезпечити розуміння того, що IT-керівники повинні приймати обґрунтовані рішення по розміщенню робочого навантаження.

Технології автоматизації, присутні не тільки в IT-інфраструктурі, але й усередині критично важливої інфраструктури енергопостачання й охолодження, допоможуть підвищити ефективність використання ресурсів і поліпшити керування активами. Спрощуйте собі життя, інвестуючи в інтелектуальні технології. Засоби, що передають інформацію про поточний стан, продуктивність і наявну ємність, існують уже досить давно, але організаціям потрібно знайти їм гідне застосування. І це повинне привести до серйозних змін. Уміння збирати й

аналізувати дані критично важливої інфраструктури в реальному часі є необхідним, але не достатньою умовою, особливо в розподіленій і різноманітної екосистеме IT.

«IT як сервіс» на підйомі

Особливо торкнемося перспективи використання в ЦОДах моделей закупівель IT як сервіс і оплати відповідно до фактичного споживання.

Стратегії закупівель в останні кілька років зрушуються убік оплати за фактичне споживання, причому спостерігається ріст цієї тенденції. Моделі закупівель на основі фактичного споживання, що забезпечують гнучкість і прозорість платежів, стають обов'язковою вимогою як фінансового директора, так і директори IT-служби. До 2027 року на IT, пропоновані як сервіс, буде доводитися до 40% витрат на корпоративну IT-інфраструктуру.

Ще однією причиною змін, що відбуваються, є цифрова трансформація. Зміна моделей робочих місць, потреба в підвищенні гнучкості IT і прозорості витрат стає застосунковим стимулом для прийняття рішень про інвестиції на основі фактичного споживання. Зі своєї сторони, постачальники послуг пропонують нові програми закупівель IT і сервісні моделі, що обіцяють застосункову вигоду в порівнянні із традиційними технологіями. Очікується, що темпи інновацій будуть і далі швидко рости.

IT-службам нові моделі обіцяють більше глибоке розуміння характеру використання й витрат, зв'язаних з усіма рівнями споживання IT. Причому нова інформація може суперечити раніше зробленим припущенням. Сервісні програми привнесуть у ЦОДи простоту й характерну хмарам гнучкість, але зажадають переходу до структури витрат на основі наявних фінансових ресурсів, що може істотно відрізнятися від більше розповсюдженої сьогодні моделі капітальних витрат.

IT-служби повинні переводити інвестиції в технологічні ресурси на прості й прозорі моделі. Необхідно розробити чіткі бізнес-параметри для всіх обслуговуючих сервісів і внесення оплати на основі фактичного використання,

для того щоб ІТ-служби могли легко оцінювати вплив технологічних витрат на результати бізнесу. Попросите постачальників технологій і сервісів пояснити, як відіб'ються їхні пропозиції, що враховують фактичне споживання, на загальній структурі витрат у короткостроковій і довгостроковій перспективі.

Програмно-конфігуруємі рішення підуть на користь периферії мережі

Приділимо особливу увагу тенденції переходу на програмно-конфігуруємі ІТ, ще раз повторивши, що в найближчі парі років більше половини своєї ІТ-інфраструктури ядра й периферії мережі компанії переведуть на програмно-конфігуруєму модель. Це дозволить домогтися бажаної гнучкості й ефективності й вплине на керування організаціями своїми ЦОДами.

Вони не будуть використовувати програмно-конфігуруєму архітектуру в якихось окремо взятих випадках, тестуя її для кожного конкретного варіанта. Мова йде про повну заміну існуючих моделей на програмно-конфігуруєму із прицілом на довгострокову перспективу. На цьому рівні програмно-конфігуруєма архітектура буде виконувати роль найважливішого прискорювача розгортання ІТ на периферії мережі. Більша частина щирих цінностей, одержуваних організаціями від програмно-конфігуруємих моделей, пов'язана з нарощуванням зусиль по розгортанню ІТ-ресурсів на периферії. Це створює більше стабільну й стандартизовану основу для початку поставок цифрових бізнес-сервісів на фабрики, у медичні установи, підприємства роздрібної торгівлі й аеропорти. Важливе значення для ІТ-служб мають і кадрові питання. Дефіцит фахівців необхідної кваліфікації буде лише наростати. Замислюючись про впровадження в себе програмно-конфігуруємої інфраструктури, компанії змушені шукати фахівців з налагодження взаємодії сервісів, автоматизації розгортання, забезпеченню закупівель, а також по оптимізації використання ресурсів.

Така оптимізація особливо важлива для компаній, що бажають витягти максимум з активів, задіяних не повною мірою.

Програмно-конфігуруєма інфраструктура не тільки спрощує розгортання ресурсів, але й оптимізує їхнє використання. Саме цього підприємствам завжди й не вистачало.

Інтелектуальні периферійні обчислення

Обчислення на периферії мережі впливають і на проектування основних ЦОДів.

Доставка IT-сервісів на периферію вимагає більшої автономії. Ресурси на периферії мережі стають більше інтелектуальними, підтримують вилучене керування й самостійно відновлюють свою працездатність після збоїв. Проводячи експерименти з технологією на периферії, компанії розраховують оснастити інтелектуальними функціями й інші ресурси ЦОД.

Периферія в цьому випадку виступає в якості поля для експериментів, покликаних підтвердити необхідність дистанційного керування. Ми переконані в тім, що інвестиції в інтелектуальну периферію – стандартизовані ресурси й інтелектуальні технології – будуть сприяти їх більше широкому поширенню й на рівні ядра. Для IT-служб це означає появу на периферії нових інструментів, що відповідають стандартам. IT-навички будуть розвиватися відповідно до цієї нової моделі, орієнтуючись в основному на поліпшення аналізу даних з метою оптимізації використання ресурсів ЦОДів.

Для закупівлі ресурсів для периферії потрібні навички в області керування даними, а також експертні знання в частині керування партнерами й договорами, якими мало хто сьогодні може похвастатися.

IDC рекомендує IT-керівникам розробити систему, що дозволить поліпшити координацію використання найважливіших мережних і IT-ресурсів. Важливе значення має й співробітництво з постачальниками, які здатні підвищити рівень прозорості й контролю. Надання такими партнерами в реальному часі даних про завантаженість, поточний стан, ефективність і витратах буде відігравати вирішальну роль при прийнятті рішень про найкращий вибір ЦОДів.

Більшості організацій прийде зайнятися перенавчанням своїх співробітників або залучати новий персонал, що вміє працювати з «цифровими близнюками» ресурсів ЦОДів.

Незважаючи на те, що стандартизовані, інтелектуальні ЦОДи будуть більше автономними, пройде ще чимало часу, перш ніж прозорість стане невід'ємною рисою всіх корпоративних ЦОДів. Число кваліфікованих фахівців, готових виконувати необхідну роботу безпосередньо на місці, досить обмежено. Думаю, що найближчим часом нас чекає бурхливий розвиток технологій доповненої реальності, які будуть підтримувати керування периферійними ресурсами.

2.2 Обґрунтування вибору засобів для побудови системи та мови програмування

Програмне забезпечення написане мовою Visual C#. Ця мова обрана виходячи з наступних міркувань. Visual C# – строго типізована об’єктно-орієнтована мова, призначена для розробки різноманітних безпечних і потужних застосунків, виконуваних у середовищі .NET Framework. Мовою Visual C# можна розробляти звичайні клієнтські застосунки Windows, веб-служби XML, розподілені компоненти, застосунки типу “сервер-клієнт”, застосунки баз даних і багато яких інших. В Visual C# є розширений редактор коду, конструктори зі зручним користувальницьким інтерфейсом, вбудований відладник і багато інших засобів, покликані спростити розробку застосунків мовою Visual C# версії 5.0 і .NET Framework версії 4.5.

Синтаксис Visual C# дуже виразний, але простий у вивченні. Усі, хто знаком з мовами C, C++ або Java з легкістю взнають синтаксис із фігурними дужками, характерний для мови Visual C#. Розроблювачі, що знають кожну із цих мов, як правило, зможуть домогтися ефективної роботи з мовою Visual C# за дуже короткий час. Синтаксис Visual C# робить простіше те, що було складно в

C++, і забезпечує потужні можливості, такі як типи значень Nullable, перерахування, делегати, лямбда-вираження й прямий доступ до пам'яті, чого немає в Java. Visual C# підтримує універсальні методи й типи, забезпечуючи більше високий рівень безпеки й продуктивності, а також ітератори, що дозволяють при реалізації колекцій класів визначати власне поводження ітерації, що може легко використовуватися в клієнтському коді. В Visual C# 5.0 вираження LINQ (Language-Integrated Query) роблять строго-типізований запит першокласною конструкцією мови.

Як об’єктно-орієнтована мова, Visual C# підтримує поняття інкапсуляції, спадкування й поліморфізму. Всі змінні й методи, включаючи метод Main – крапку входу застосунка – інкапсулюється у визначення класів. Клас може успадковувати безпосередньо з одного родового класу, але може реалізовувати будь-яке число інтерфейсів. Для методів, які перевизначають віртуальні методи в батьківському класі, необхідно ключове слово override, щоб виключити випадкове повторне визначення. У мові Visual C# структура схожа на полегшений клас: це тип, що розподіляється по стопках, що реалізує інтерфейси, але не підтримуюче спадкування.

На застосунок до основних описаних об’єктно-орієнтованих принципів, мова Visual C# спрощує розробку компонентів програмного забезпечення завдяки декільком інноваційним конструкціям мови, у число яких входять наступні:

– Інкапсульовані підписи методів, називані делегатами, які підтримують строго-типізовані повідомлення про події.

– Властивості, що виступають у ролі методів доступу для закритих змінних-членів.

– Атрибути з декларативними метаданими про типи під час виконання.

– Вбудовані коментарі XML-документації.

– LINQ (Language-Integrated Query), що пропонує вбудовані можливості запитів у різних джерелах даних.

Якщо буде потрібно забезпечити взаємодію з іншим програмним забезпеченням Windows, таким як об'єкти COM або власні бібліотеки DLL Win32, у мові Visual C# можна використовувати процес, що називається "Interop". Процес Interop дозволяє програмам на Visual C# виконувати практично будь-які дії, які може виконувати вихідний застосунок на C++. Мова Visual C# підтримує навіть покажчики й поняття "небезпечного" коду для тих випадків, коли прямий доступ до пам'яті має вкрай важливе значення.

Процес побудови Visual C# у порівнянні з C і C++ простий і є більше гнучким, чим в Java. Немає окремих файлів заголовка, а методи й типи не потрібно повідомляти в певному порядку. У вихідному файлі Visual C# може бути визначене будь-яке число класів, структур, інтерфейсів і подій.

Архітектура платформи .NET Framework

Програма мовою Visual C# виконується в середовищі .NET Framework – інтегрованому компоненті Windows, що містить віртуальну систему виконання (середовище CLR) і уніфікований набір бібліотек класів. Середовище CLR являє собою комерційну реалізацію корпорацією Майкрософт інфраструктури CLI, що є міжнародним стандартом, який лежить в основі створення середовищ виконання й розробки, у яких забезпечується тісна взаємодія між мовами й бібліотеками.

Вихідний код, написаний мовою Visual C#, компілюється в проміжну мову (IL) у відповідності зі специфікацією CLI. Код IL і ресурси, такі як растрові зображення й рядки, зберігаються на диску у файлі, що виконується, називаному складанням, з розширенням EXE або DLL у більшості випадків. Складання містить маніфест із відомостями про типи складання, версії, мови й регіональні параметри та вимоги безпеки.

При виконанні програми на Visual C# складання завантажується в середовище CLR залежно від відомостей у маніфесті. Далі, якщо вимоги безпеки дотримані, середовище CLR виконує JIT-компіляцію для перетворення коду IL в інструкції машинного коду. Середовище CLR також надає інші служби, що

відносяться до автоматичного збору сміття, обробки виключень і керуванню ресурсами. Код, виконуваний середовищем CLR, іноді називають "керованим кодом" у протиставлення "некерованому коду", що компілюється в машинний код, призначений для певної системи. Далі показані відносини під час компіляції й час виконання між файлами з вихідним кодом Visual C#, бібліотеками класів .NET Framework, складаннями й середовищем CLR.

Взаємодія між мовами є ключовою особливістю .NET Framework. Оскільки код IL, створюваний компілятором Visual C# відповідає специфікації CTS, код IL на основі Visual C# може взаємодіяти з кодом, створюваним версіями мов Visual Basic, Visual C++, Visual J# платформи .NET Framework і ще більш ніж 20 CTS-сумісних мов. В одному складанні може бути кілька модулів, написаних на різних мовах платформи .NET Framework, і типи можуть посилатися один на одного, як якби вони були написані на одній мові.

Крім служб часу виконання, в.NET Framework також є велика бібліотека, що складається з більш ніж 4000 класів, організованих по просторах імен, які забезпечують різноманітні корисні функції для будь-яких дій, починаючи від введення й виведення файлів для керуванням рядками для розбивки XML, і закінчуючи елементами керування Windows Forms. У звичайному застосунку мовою Visual C# бібліотека класів .NET Framework інтенсивно використовується для "устрою" коду.

2.3 Розгорнута постановка завдання

Згідно з технічним завданням на випускну кваліфікаційну роботу за першим (бакалаврським) рівнем вищої освіти, реалізації підлягає програмне забезпечення, яке призначено для системи інтелектуального підвищення відказостійкості периферійної ЦОД.

В процесі розробки випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти необхідно виконати наступний обсяг роботи:

а) провести аналіз існуючих систем-аналогів для виявлення їх позитивних і негативних якостей. Результати аналізу врахувати в подальших розробках;

б) вибрати та обґрунтувати методику побудови системи контролю роботи технологічного обладнання на виробництві в автоматизованому режимі. Розробити функціональну та структурну схеми системи;

в) розробити програмне забезпечення системи, що дозволить реалізувати поставлену технічним завданням задачу. Побудувати блок-схеми алгоритмів програми та підпрограми;

г) організувати інтерфейс користувача з метою формування та виводу на екран ЕОМ повідомлень про некоректні дії користувача та нестандартні ситуації в роботі технологічного обладнання;

д) розробити рекомендації по організаційних та методичних заходах, які забезпечать впровадження системи в промислову експлуатацію та її подальшу успішну експлуатацію;

е) провести розрахунки по визначенню економічної ефективності розробленої системи;

ж) розробити заходи по охороні праці при впровадженні та експлуатації системи, а також розробити заходи з цивільного захисту;

з) сформувати висновки про виконаний обсяг робіт та одержані результати.

3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

3.1 Опис функціонування системи

Багато підприємств, переходячи в хмару або на зовнішнє розміщення, приділяють недостатню увагу стійкам, що залишилися, віддаючи пріоритет доступності великих ЦОД. Така логіка містить вади, адже в більшості випадків працююче на місцях устаткування відповідає за виконання настільки ж або навіть більше важливих завдань, чим ті, які переведені на хмарні ресурси.

Що звичайно залишається в компанії? По-перше, пропрієтарні, важливі для бізнесу застосунки, а по-друге, мережне встаткування для підключення до хмари. Якими можуть бути наслідку при виникненні проблем з доступом до застосунків? Якщо припустити, що в офісі компанії залишається працювати то ж кількість співробітників, що й раніше, але число стійок скоротилося до пари штук, важливість кожної з них зростає. Локально розташоване встаткування конче потрібно для забезпечення зв'язку з бізнес-застосунками, використовуваними в повсякденній діяльності. З обліком того, що усе більше й більше ресурсів переміщається в хмару, від надійності й швидкості підключення до них залежить продуктивність співробітників.

Все це свідчить про необхідність зміни принципів проектування малих ЦОД. Не можна концентруватися тільки на централізованих і регіональних центрах обробки даних, необхідно приділяти увагу й локальні площадки, оскільки саме вони є найбільш слабкою ланкою. Далі будуть описані кращі практики, які необхідно використовувати для забезпечення високої продуктивності бізнесу, що володіє безліччю зв'язків.

Більше повні метрики доступності

При аналізі гібридного середовища, у якій усе компоненти взаємозалежні, виникає важливе питання: чи належні ми переглянути наш підхід до оцінки її критичності й резервування?

Інструменти, які галузь використовує сьогодні, спрямовані на те, щоб зробити окремий ЦОД максимально надійним. При проектуванні конкретних площадок стандарти Uptime Institute ураховуються таким чином, щоб досягти необхідного рівня доступності («кількості дев'яток»). Відмова звичайно визначається як порушення роботи IT-устаткування.

Використовувані інструменти й метрики не враховують не тільки залежність від кількості ЦОД і числа користувачів, але й критичність функцій, на які вплинула дана несправність, або відказостійкость застосунка (ПЗ). Ми вважаємо, що все це важливо для руху вперед.

Зміна очікуваного рівня доступності. Очікування молодих співробітників і представників старшого покоління розрізняються. У міру старіння останніх все більшу вагу здобувають мілленіали, а разом з ними міняються й очікування. Люди, що народилися після 1980-го року, росли з думкою про те, що треба бути «завжди на зв'язку, завжди онлайн», тобто всі прилади й пристрої ніколи не повинні вимикатися. У них нульова терпимість до перебоїв у наданні сервісів. Фактично 82% мілленіалів уважають, що вибір нового місця роботи буде залежати від рівня застосовуваних у компанії технологій.

Якщо припустити, що ця тенденція буде розвиватися, надзвичайно важливо знайти більше універсальні способи моніторингу відказостійкості ЦОД, за допомогою яких можна було б одержати вичерпні дані для внесення необхідних змін у їхній дизайн. Як учить стара приказка, «не можна управляти тим, що не можна виміряти». Метрики відказостійкості необхідно вдосконалити, щоб вони відповідали вимогам сучасного бізнесу.

Зміна підходу. Зміна точки зору на доступність може привести до іншої стратегії розвитку.

Як приклад приведемо комунальні підприємства (енергопостачання) і їхнє відношення до доступності сервісів. Вони не тільки стежать за технічним станом своїх електростанцій і високовольтних ліній (метафорично їх «центральним ЦОД»), але й виконують великий обсяг суміжних робіт, наприклад підріжуть вітки дерев і ремонтують трансформаторні підстанції (їх «периферійні ЦОД»). Успіх всіх цих заходів оцінюється з урахуванням стабільності і якості електрики, що поставляється користувачам. Галузі ЦОД необхідно рухатися до описаної моделі, у якій периферійна частина важлива так само, як і центральна частина.

Доступність двох систем з обліком того, що бізнес залежить від доступу до обох, обчислюється по формулі:

$$\text{Доступність}_{\text{системи}} = \text{Доступність}_1 * \text{Доступність}_2$$

Дана формула розрахунку доступності ЦОД припускає, що ефективність роботи користувача залежить від доступності й продуктивності локального й центрального ЦОД. Якщо, наприклад, центральна площадка доступна 99,98% часу (ЦОД Tier III, 1,6 год простою в рік), а локальна площадка – 99,67% (ЦОД Tier 1, 28,8 год простою), загальний час простою з погляду користувача складе 99,98% * 99,67% = 99,65% (30,7 год простою).

Як можна оцінити вплив всієї екосистеми центрів обробки даних на продуктивність бізнесу й зв’язність усередині системи?

Не всі центри обробки даних мають однаковий вплив на бізнес. Вирішальний фактор – кількість співробітників, що обслуговуються. Наприклад, локальний ЦОД на 1000 чоловік може мати істотно більше значення, чим ЦОД на 10 чоловік.

Загальний час простою залежить головним чином від кількості периферійних площадок Tier 1: чим їх більше, тим менше число годин, коли працюють всі площадки.

У міру збільшення числа ЦОД з різним рівнем доступності й кількості робочих місць, що обслуговуються ними, порахувати доступність стає набагато складніше. Крім того, зазначена формула не є повної, тому що вона не враховує

рейтингу виконуваною кожною площадкою бізнес-функції. Обслуговування клієнтів або виробництва завжди важливіше, ніж підтримка адміністративного персоналу, що у випадку відмови мережі може працювати дистанційно.

Ми вважаємо, що кращим підходом до цілісної оцінки всіх площадок є використання карт показників. Це допоможе виявити найбільш важливі для діяльності підприємства площадки, які мають потребу в першочерговій модернізації. Карта показників містить інформацію про доступність і відповідний час простою кожної площадки в гібридному середовищі ЦОД (ідеальні показники), а також і, що істотніше всього, рівень її критичності для бізнесу. У випадку із ЦОД інтенсивність наслідків відмови кожної площадки залежить:

– від кількості робочих місць, що постраждали від збою систем;

– виконуваної функції.

Часто для оцінки використовується шкала від 1 до 5, де 1 – це найменший вплив на бізнес, а 5 – найбільше. Зверніть увагу на те, що компанії з різних галузей економіки будуть по-різному оцінювати важливість аналогічних площадок. Ключ до успіху – послідовний підхід до оцінки всіх наявних площадок.

У даному прикладі розглядаються п'ять центрів обробки даних, що утворять гіпотетичну екосистему. Щорічний час простою кожного з них множиться на задану величину «серйозності наслідків відмови» для одержання зваженого значення.

Площадки можна просто відсортувати за цим значенням: найбільше вказує на найбільший пріоритет з погляду потреби в поліпшенні. Із цією же метою можна скласти рейтинг площадок на підставі процентного показника оцінки (як показано в прикладі, «вплив сайту на оцінку»).

Дана процедура припускає послідовні ітерації. Як тільки рівень доступності площадки 4 у прикладі підвищиться, нова площадка виявиться на початку списку як найважливіша. Протягом цього безперервного циклу

поліпшень будуть модернізовані площадки, відмова яких може мати найбільші наслідки.

При правильному підході до оцінки доступності можна явно виділити площадки, де поліпшення дадуть найбільший приріст продуктивності й економічний ефект. У більшості випадків після виконання подібних розрахунків стає ясно, що периферійні ЦОД, що часто мають більше низький рівень доступності, впливають на бізнес.

Кращі практики на периферії

При використанні правильних метрик і методів стає ясно, що інфраструктуру периферійних ЦОД необхідно переосмислити. Стандартні підходи до їхнього проектування (як було описано раніше) не відповідають ступеню важливості даних площадок для бізнесу. Поліпшення потрібні в наступних областях:

– фізична безпека;

– моніторинг (DCIM), обслуговування, дистанційний контроль;

– резервні системи живлення й охолодження;

– резервування каналів зв'язку.

Безпечне робітниче середовище

Невеликі локальні ЦОД нерідко розміщаються в приміщенні з відкритим доступом (наприклад, в офісі, де перебуває кілька відділів). Найчастіше окреме приміщення для стійок не виділяється, тому вони залишаються відкритими, а виходить, і не захищеними від будь-яких випадкових або навмисних впливів.

Кращими практиками для зниження даних ризиків є:

– перенос устаткування в приміщення, що замикається, або спеціалізовану шафу;

– забезпечення біометричного або іншого контролю доступу;

– у випадку несприятливих умов, розміщення встаткування в шафі, оснащеної засобами захисту від пожеж, повеней, вологості, вандалізму, впливів електромагнітного випромінювання;

– розгортання системи цілодобового моніторингу навколишнього середовища, а також системи відеоспостереження.

Керування ЦОД

На різних периферійних площадках процедури керування й експлуатації нерідко розрізняються (якщо такі процедури визначені). На керування сотнями або тисячами периферійних площадок витрачається чимало часу й грошей. Крім того, на багатьох площадках рівень доступності залежить від використання суміжної інфраструктури на об'єкті (генераторів, комутаційних пристроїв, систем охолодження.

Кращими практиками для зниження даних ризиків є:

– аналіз застосовуваних методів і систем керування;

– створення єдиної системи моніторингу для всієї інфраструктури на всіх наявних площадках;

– розгортання системи дистанційного контролю, якщо ресурси обмежені.

Живлення й охолодження

Інфраструктура систем живлення й охолодження (наприклад, ІБП і кондиціонери) на периферійних площадках нерідко не передбачає резервування. Це приводить до критичних відмов, а також до неможливості обслуговувати системи, не припиняючи роботу ЦОД. У деяких випадках у приміщеннях відсутня спеціальна система охолодження, що приводить до перегріву встаткування. Частина інженерних систем нерідко використовується декількома компаніями усередині одного будинку, тому рівень доступу до ЦОД залежить від доступності цих ресурсів.

Кращими практиками для зниження даних ризиків є:

– вимір температури й вологості для оцінки необхідного рівня охолодження (пасивне охолодження, активне охолодження або спеціальна система охолодження);

– виділення резервних ліній живлення для організації паралельного обслуговування на найбільш важливих площадках;

– підключення самих критичних ланцюгів до резервного генератора.

Доступність мережі

Як було описано раніше, підключення до хмари є ключовою умовою функціонування периферійних площадок. Однак часто буває, що з'єднання надається тільки одним провайдером. Ця обставина створює єдину крапку відмови. Крім того, безладдя в кабельному господарстві може стати причиною людських помилок.

Кращими практиками для зниження даних ризиків є:

– підключення резервного каналу передачі даних;

– використання засобів організації кабелів (кабельні канали, організатори, фіксатори й т.д.);

– маркування й колірне кодування проводів.

Розвиток хмарних технологій приводить до того, що керівники компаній всі частіше починають замислюватися про необхідність створення гібридних середовищ для хмарних і локальних ЦОД (на периферії). Незважаючи на те що кількість устаткування, що залишилося, зменшується, його значення для бізнесу навіть зростає. Причини наступні:

– оскільки усе більше застосунків переноситься в хмару, підключення до хмари стає найважливішим фактором успішної роботи підприємства;

– співробітники, що звикли до того, що онлайн-сервіси завжди доступні, нетерпимо ставляться до їхніх простоїв.

На жаль, сьогодні дизайн більшості периферійних ЦОД має вади, що приводить до дорогих простоїв. Необхідний систематичний підхід до оцінки рівня доступності ЦОД у гібридному середовищі, що дозволить забезпечити найбільшу економічну ефективність інвестицій.

Представлена концепція використання карт показників дозволяє одержати цілісну картину стану робітничого середовища з урахуванням числа користувачів, що обслуговуються кожним ЦОД, і критичності кожної площадки

для функціонування бізнесу. Цей метод допомагає зрозуміти, куди необхідно інвестувати засобу в першу чергу.

МікроЦОД високої заводської готовності являють собою найбільш простий спосіб забезпечення безпечного й високодоступного інженерного середовища на периферії. Кращі практики, такі як використання резервних ІБП і захищених стійок, системи організації кабелів і повітряних потоків, вилучений моніторинг і резервні канали передачі даних, дозволяють гарантувати необхідний рівень доступності для найбільш важливих площадок.

3.2 Розробка структурної схеми

У даній роботі для системи інтелектуального підвищення відказостійкості периферійної ЦОД пропонується використовувати процес кодування та декодування інформації за допомогою циклічного кодування.

Структурна схема системи підвищення надійності зберігання даних на носіях інформації периферійної ЦОД за допомогою кодеку циклічного коду зображена на рисунку 3.1.

З цієї схеми ми бачимо, що над вхідними даними, перед записом на носіях інформації периферійної ЦОД, відбуваються перетворення кодеком циклічного коду. Після кодування дані записуються на носій периферійної ЦОД.

При зчитуванні інформації, розроблене програмне забезпечення декодує інформацію, яка зберігається на носієві периферійної ЦОД, і якщо потрібно, після проведення відповідних перевірок, проводить відновлення втраченої інформації. Якщо таке відновлення неможливе, то програма видає відповідне повідомлення.

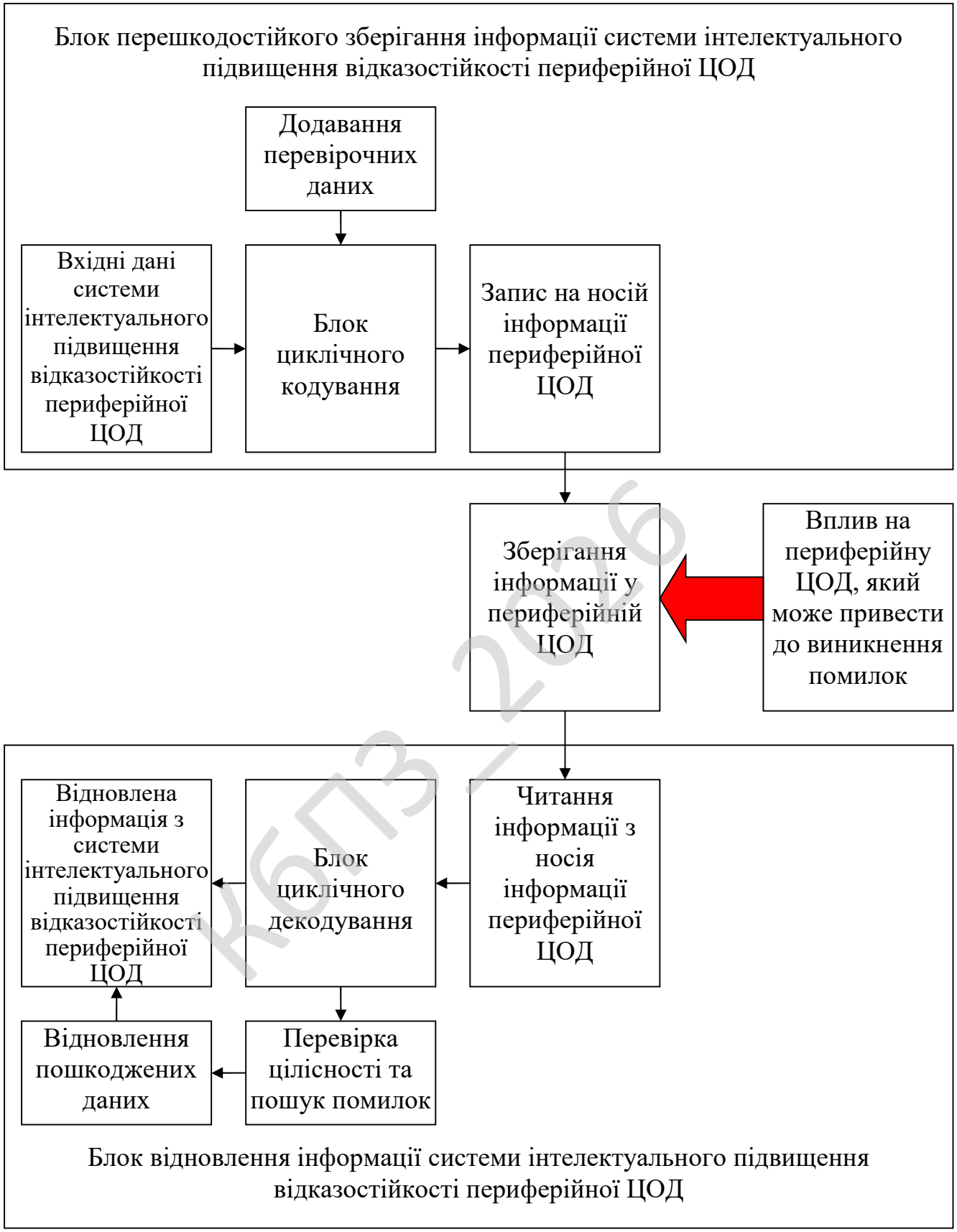

Рисунок 3.1 – Структурна схема системи

Відказостійкість периферійної ЦОД з урахування енергоспоживання

За своєю природою, центри обробки даних побудовані на ідеї, що електроенергія буде доступна завжди. Однак робочі навантаження штучного інтелекту продовжують зростати, а щільність стійок зростає швидше, ніж можна ефективно керувати. У 2026 році та в майбутньому новою основою стійкості є інтегроване середовище розподілу електроенергії, зокрема модульні електричні кімнати, які об'єднують розподільчі пристрої, ДБЖ, акумуляторні системи, трансформатори, PDU, шинопроводи, системи керування та програмне забезпечення в одну цілісну, попередньо спроектовану систему.

Глобальний попит на електроенергію з боку центрів обробки даних різко зросте в цьому десятилітті через навантаження на навчання та логічний висновок штучного інтелекту, які споживають величезні та стабільні обсяги енергії. Водночас затримки підключення до мережі, обмеження на землю, порушення ланцюга поставок та посилення норм ESG створюють зростаючий тиск на способи проектування, розгортання та експлуатації енергетичної інфраструктури. На перетині цих тенденцій роль обладнання для розподілу електроенергії фундаментально переосмислюється.

Традиційно резервне живлення в центрі обробки даних розглядалося як щось на зразок страхового поліса: критично важливе, але здебільшого непомітне, доки щось не пішло не так. Сьогодні такого мислення вже недостатньо. Середовища штучного інтелекту високої щільності створюють більше навантаження на електричні системи, при цьому коливання потужності, тимчасові збої та швидкі зміни навантаження стають все більш поширеними. Навіть короткі перерви можуть мати непропорційний вплив, особливо для навчальних навантажень штучного інтелекту, що виконуються у великих масштабах протягом тривалих періодів часу.

У цьому новому контексті стійкість більше не може бути прикріпленою; вона має бути вбудована в фундамент як частина єдиної системи. Джерело безперебійного живлення (ДБЖ) залишається важливим компонентом, але лише

як частина повністю інтегрованого модульного електричного приміщення, яке координує, захищає, контролює та розподіляє живлення від початку до кінця.

Цей зсув стає ще більш важливим, оскільки центри обробки даних розширюються в нетрадиційні місця зі слабкішою стійкістю мережі та більшою схильністю до перебоїв. У цих середовищах модульне електричне приміщення стає операційною основою, забезпечуючи надійність, видимість та контрольованість усього ланцюга живлення, від мережі до стійки.

Проектування профілів робочого навантаження на основі штучного інтелекту

Штучний інтелект змінює споживання енергії центрами обробки даних та способи її використання. На відміну від передбачуваних корпоративних або хмарних робочих навантажень, навчання роботі зі штучним інтелектом може передбачати тривалі періоди високого використання, зі швидкими циклами нарощування та зниження потужності. Ці закономірності створюють нові навантаження на всю електричну екосистему, чітко показуючи, що стійкість в епоху штучного інтелекту не може бути досягнута, зосереджуючись на окремих компонентах або просто збільшуючи розміри інфраструктури. Це вимагає системного підходу, який передбачає поведінку штучного інтелекту та забезпечує безперебійну роботу кожної частини енергетичного ланцюга.

Саме тут модульні електроцентралі стають у пригоді. Замість того, щоб розглядати ДБЖ, розподільчі пристрої, акумулятори, елементи керування та моніторинг як окремі активи, модульні електроцентралі інтегрують їх в єдине середовище, розроблене для стабільності за умов коливань та високої щільності робочих навантажень. Об'єднуючи критично важливі елементи в одну скоординовану систему, вони забезпечують плавніші переходи між джерелами живлення, кращий захист від перехідних процесів та надійнішу основу для тривалих навчальних сеансів ШІ.

Модуляризація як відповідь на швидкість та складність

Одним із найважливіших зрушень у сфері енергоінфраструктур центрів обробки даних є перехід до модульних рішень, що виготовляються поза межами об'єкта. Збірні силові модулі, як на шасі, так і в контейнерах, дозволяють проектувати, збирати та тестувати системи ДБЖ, акумулятори, розподільчі пристрої та контрольне обладнання в контрольованих заводських умовах. Такий підхід скорочує час монтажу на місці та зменшує нестачу робочої сили, одночасно покращуючи загальну якість.

Для операторів центрів обробки даних, які стикаються з тривалими термінами підключення до мережі або графіками розгортання, модульні системи живлення забезпечують паралельну роботу. Поки зали обробки даних будуються, інфраструктуру розподілу живлення можна виготовити та ввести в експлуатацію поза межами об'єкта, готову до швидкої інтеграції. Результатом є швидший час введення в експлуатацію та зниження ризиків проекту.

Модульна конструкція також забезпечує гнучкість. Зі зростанням робочих навантажень штучного інтелекту та збільшенням щільності потужності, модульні електричні системи можна масштабувати легше, ніж стаціонарні установки, що бачили раніше. Така адаптивність є більш цінною в середовищі, де довгострокові потреби в електроенергії важко передбачити з певністю.

Стійкість у світі з обмеженнями ESG

Стійкість більше не можна розглядати окремо від сталого розвитку. Регулятори та громади приділяють більше уваги часу роботи генераторів, викидам, зберіганню палива та загальній енергоефективності. Водночас оператори центрів обробки даних перебувають під тиском, щоб продемонструвати переконливий прогрес у досягненні цільових показників нульового рівня викидів.

Це створює делікатний баланс; модульні електричні кімнати повинні забезпечувати абсолютну надійність, але водночас вони повинні відповідати постійно зростаючим очікуванням ESG, приділяючи більше уваги ефективності,

продуктивності протягом життєвого циклу та інтелектуальному моніторингу. Високоефективні конструкції зменшують втрати під час нормальної роботи, тоді як розширений моніторинг дозволяє проводити прогнозне обслуговування, продовжуючи термін служби активів та зменшуючи кількість відходів.

Для резервного виробництва енергії гібридні підходи набирають обертів. Інтеграція генераторів із системами ДБЖ, накопичувачами енергії та, де це можливо, відновлюваними джерелами енергії дозволяє операторам зменшити залежність від викопного палива без шкоди для стійкості. Хоча повністю відновлюване резервне живлення залишається довгостроковою метою, поступові вдосконалення в проектуванні та управлінні системою можуть забезпечити значні екологічні переваги вже сьогодні.

Важливість інтеграції та експертизи

Оскільки системи розподілу електроенергії стають складнішими та відіграють центральну роль у продуктивності центрів обробки даних, важливо враховувати важливість інтеграції. Фрагментовані підходи, коли різні елементи ланцюга живлення проектуються та постачаються ізольовано, збільшують ризик несумісності, затримок та експлуатаційних проблем.

Повністю інтегрований підхід, що охоплює проектування, виробництво, випробування та введення в експлуатацію, забезпечує більшу впевненість у продуктивності та термінах. Він також дозволяє точно адаптувати енергетичну інфраструктуру до експлуатаційних потреб центру обробки даних, а не змушувати операторів адаптувати свої об'єкти до готових рішень.

Проектування енергетичних систем для середовищ, які не терплять збоїв, вимагає глибоких інженерних знань, розуміння реальних умов експлуатації та здатності передбачати, як системи розвиватимуться з часом. Оскільки штучний інтелект продовжує змінювати ландшафт центрів обробки даних, цей спеціалізований досвід буде ключовою відмінністю.

Наступне покоління центрів обробки даних визначатиметься не лише кількістю споживаної енергії, а й тим, наскільки розумно та стійко вони нею

керують. Системи розподілу електроенергії виходять з фону на передній план у проектуванні центрів обробки даних, відіграючи центральну роль у забезпеченні зростання в умовах невизначеності.

Для операторів виклик очевидний: стійкість більше не може бути фіксованою. Вона має бути вбудована з самого початку, а системи розподілу електроенергії мають розглядатися як стратегічна інфраструктура, а не як допоміжне обладнання. Ті, хто дотримується модульності, інтеграції та перспективного проектування, будуть найкраще підготовлені до подолання енергетичних викликів епохи штучного інтелекту та забезпечення роботи критично важливої цифрової інфраструктури, незалежно від того, що їй кидає мережа.

3.3 Розробка функціональної схеми

На рисунку 3.2 зображена функціональна схема системи. З неї бачимо, що розроблена система підвищення стійкості до уражень інформації яка записана на дисках складається з наступних основних функціональних блоків:

– сам носій інформації периферійної ЦОД;

– кодер циклічного коду, який використовується, коли відбувається запис інформації на носій інформації периферійної ЦОД, при цьому необхідно враховувати, що об’єм інформації, яка записується на носій інформації периферійної ЦОД, повинна бути меншою, ніж об’єм носія інформації периферійної ЦОД, в зв’язку, з тим, що коди циклічного коду відносяться до кодів з надмірністю, за рахунок якої й відбувається кодування;

– декодер циклічного коду, який використовується при читанні даних с відповідного носія інформації периферійної ЦОД;

– файли для перешкодостійкого збереження

– відновлені файли.

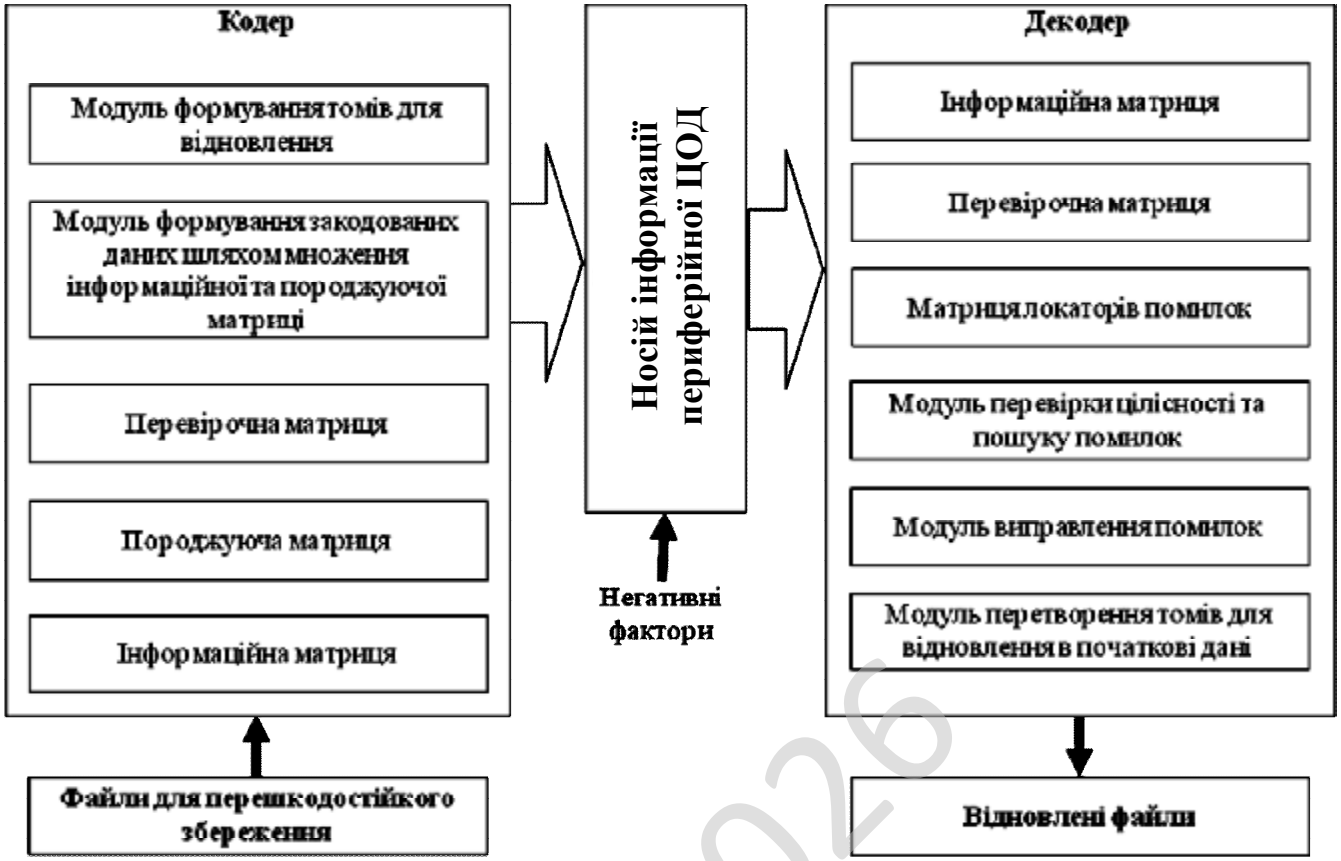

Рисунок 3.2 – Функціональна схема системи

Розглянемо більш детально перераховані функціональні блоки кодування та декодування за допомогою циклічного кодування.

Блок кодування складається з наступних підблоків:

– Модуль формування томів для відновлення.

– Модуль формування закодованих даних шляхом множення інформаційної та породжуючої матриці.

– Перевірочна матриця.

– Породжуюча матриця.

– Інформаційна матриця.

Блок декодування складається з наступних підблоків:

– Інформаційна матриця.

– Перевірочна матриця.

– Матриця локаторів помилок.

– Модуль перевірки цілісності та пошуку помилок.

– Модуль виправлення помилок.

– Модуль перетворення томів для відновлення в початкові дані.

Коди циклічного коду базуються на спеціальному розділі математики – полях Галуа (GF) або кінцевих полях. Арифметичні дії (+, – , х, / і т.д.) над елементами кінцевого поля дають результат, що також є елементом цього поля. Кодер та декодер циклічного коду повинні вміти виконувати ці арифметичні операції. Ці операції для своєї реалізації вимагають спеціального устаткування або спеціалізованого програмного забезпечення.

Кодове слово циклічного коду формується із залученням спеціального полінома. Всі коректні кодові слова повинні ділитися без залишку на ці утворюючі поліноми. Загальна форма утворюючого полінома має вигляд:

$$g(x) = (x\text{-}a^{i})(x\text{-}ai^{+1})\ldots(x\text{-}a^{i+2t}), \quad (3.1)$$

а кодове слово формується за допомогою операції:

$$c(x) = g(x)i(x), \quad (3.2)$$

де – $g(x)$ є утворюючим поліномом;

– $i(x)$ являє собою інформаційний блок;

– $c(x)$ – кодове слово, що називається простим елементом поля.

$2t$ символів парності в кодовому слові циклічного коду визначаються з наступного співвідношення:

$$p(x) = i(x)\cdot x^{n-k} \bmod g(x). \quad (3.3)$$

Перейдемо до розгляду іншого функціонального блоку – декодеру циклічного коду.

Декодер працює наступним чином.

Введемо позначення:

– $r(x)$ – Отримане кодове слово.

– S_i – Синдроми.

– $L(x)$ – Поліном локації помилок.

– X_i – Положення помилок.

– Y_i – Значення помилок.

– $c(x)$ – Відновлене кодове слово.

– v – Число помилок.

Отримане кодове слово r(x) являє собою вихідне (передане) кодове слово c(x) плюс помилки: $r(x) = c(x) + e(x)$.

Декодер циклічного коду намагається визначити позицію й значення помилки для числа t помилок (або $2t$ втрат) і виправити помилки й втрати.

Обчислення синдрому

Обчислення синдрому схоже на обчислення парності. Кодове слово циклічного коду має $2t$ синдромів, це залежить тільки від помилок (а не переданих кодових слів). Синдроми можуть бути обчислені шляхом підстановки $2t$ коріння утворюючого полінома $g(x)$ в $r(x)$.

Знаходження позицій символьних помилок

Це робиться шляхом рішення системи рівнянь із t невідомими. Існує кілька швидких алгоритмів для рішення цього завдання. Ці алгоритми використовують особливості структури матриці кодів циклічного коду й сильно скорочують необхідну обчислювальну потужність. Робиться це у два етапи:

1. Визначення полінома локації помилок

Це може бути зроблене за допомогою алгоритму Berlekamp-Massey або алгоритму Евкліда. Алгоритм Евкліда використовується частіше на практиці, тому що його легше реалізувати, однак, алгоритм Berlekamp-Massey дозволяє одержати більш ефективну реалізацію встаткування й програм.

2. Знаходження кореня цього полінома. Це робиться із залученням алгоритму пошуку Chien.

Знаходження значень символьних помилок

Тут також потрібно вирішити систему рівнянь із t невідомими. Для рішення використовується швидкий алгоритм Forney.

Розглянувши усі блоки функціональної схеми перейдемо до розгляду діаграми взаємодії процесів, які відбуваються у системі.

3.4 Розробка діаграми процесів

Розглянемо розроблену діаграму процесів яка зображена на рисунку 3.3. Основна будова діаграми процесів полягає у графічному представленні складу сукупностей даних, що характеризуються як співвідношення різних частин кожної з сукупностей.

Склад статистичної сукупності графічно може бути представлений як за допомогою абсолютних, так і відносних показників.

Графічне зображення складу сукупності по абсолютними і відносними показниками сприяє проведенню більш глибокого аналізу і дозволяє проводити аналіз системи.

Діаграма взаємодії процесів використовується для візуалізації процесів обробки даних (структурне проектування).

Для розробника вважається звичним спочатку креслити діаграму взаємодії процесів даних рівня контексту, завдяки чому буде показано взаємодію системи.

Ця діаграма в подальшому підлягає уточненню шляхом деталізації процесів та потоків даних з метою показати систему що розробляється.

При детальному її розгляді можна побачити як саме проходе взаємодія у розробленій системі.

Використовується модель проектування, графічне представлення «потоків» даних в інформаційній системі.

Діаграми потоків даних містять чотири типи елементів:

– Зовнішні по відношенню до системи сутності.

– Потоки даних між елементами трьох попередніх типів.

– Процеси які являють собою трансформацію даних в рамках описуваної системи.

– Сховища даних (репозиторії).

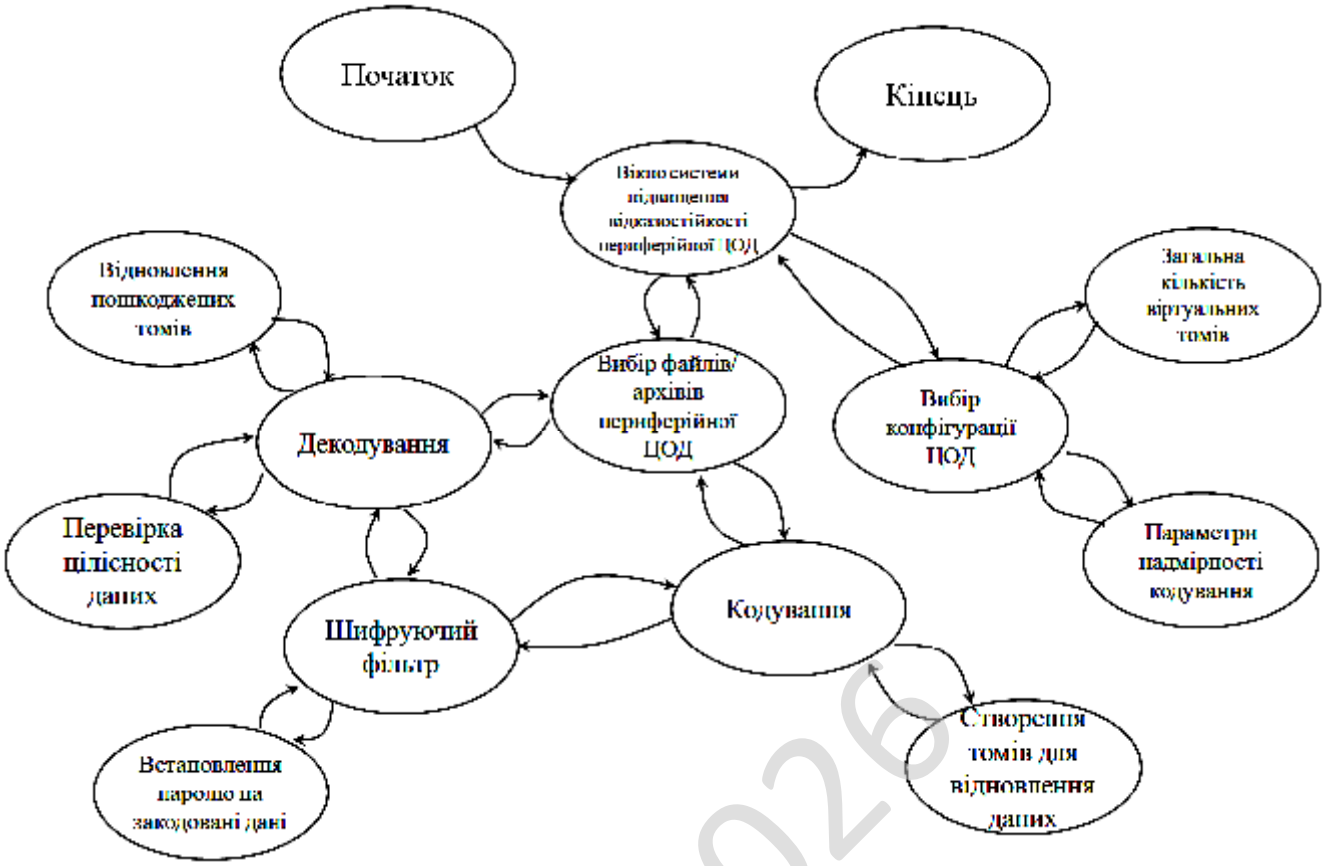

Рисунок 3.3 – Діаграма взаємодії процесів

Таким чином, розглянувши опис системи, структурну, функціональну схеми системи, та діаграму взаємодії процесів перейдемо до опису блок-схем основної програми, та підпрограм, які використовуються, для реалізації системи.

4 РЕАЛІЗАЦІЯ ПРОЕКТУ. РОЗРАХУНКИ І ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ПРАВИЛЬНІСТЬ ПРОЕКТНИХ РІШЕНЬ

4.1 Блок-схеми та опис алгоритмів функціонування системи

Розглянемо реалізацію бакалаврської дипломної роботи. Були проведені розрахунки і підібрані набори тестових даних для перевірки правильності реалізації проектних рішень.

Було створено блок-схеми роботи системи. Перед їх розглядом необхідно провести роз’яснення який саме тип блок-схем використовується. Блок-схема це представлення задачі для її аналізу або розв'язування за допомогою спеціальних символів (геометричних образів), які позначають такі елементи, як операції, потік, дані тощо.

Блок вхідних та вихідних даних прийнято позначати паралелограмом, блок обчислень (обробки) даних – прямокутником, блок прийняття рішень – ромбом, еліпсом – початок та кінець алгоритму.

У інформаційних технологіях функціональна схема складається з функціональних блоків, які являють собою конструктивно відособлені частини (елементи або пристрої) автоматичних систем, які виконують певні функції. Функціональні блоки на схемі позначають прямокутниками, всередині яких надписують їх найменування відповідно до функцій, що виконуються. Зв'язки між функціональними блоками (внутрішні впливи) позначаються лініями зі стрілками, які вказують напрям впливів.

Функціональні схеми можуть виконуватися в укрупненому і розгорненому вигляді. У першому випадку на схемі зображають найважливіші блоки системи і зв'язки між ними.

У другому варіанті схема відображається більш детально, що полегшує її читання та ілюструє принцип роботи.

Основні елементи схем алгоритму це термінатор, процес, рішення, зумовлений процес (підпрограма), дані та з'єднувач. Термінатор це елемент відображає вхід із зовнішнього середовища або вихід з неї (найчастіше застосування – початок і кінець програми). Всередині фігури записується відповідна дія.

Процес це виконання однієї або кількох операцій, обробка даних будь-якого виду (зміна значення даних, форми подання, розташування). Всередині фігури записують безпосередньо самі операції. Рішення це показує рішення або функцію перемикального типу з одним входом і двома або більше альтернативними виходами, з яких тільки один може бути обраний після обчислення умов, визначених всередині цього елементу.

Вхід в елемент позначається лінією, що входить зазвичай у верхню вершину елементу. Якщо виходів два чи три то зазвичай кожен вихід позначається лінією, що виходить з решти вершин (бічних і нижній). Якщо виходів більше трьох, то їх слід показувати однією лінією, що виходить з вершини (частіше нижній) елемента, яка потім розгалужується.

Відповідні результати обчислень можуть записуватися поруч з лініями, що відображають ці шляхи.

Зумовлений процес (підпрограма) це символ відображає виконання процесу, що складається з однієї або кількох операцій, що визначені в іншому місці програми (у підпрограмі, модулі). Всередині символу записується назва процесу і передані в нього дані.

Дані це перетворення у форму, придатну для обробки (введення) або відображення результатів обробки (виведення). Цей символ не визначає носія даних (для вказівки типу носія даних використовуються специфічні символи). З'єднувач це символ відображає вихід в частину схеми і вхід з іншої частини цієї схеми.

Використовується для обриву лінії та продовження її в іншому місці (приклад: поділ блок-схеми, що не поміщається на листі). Відповідні сполучні символи повинні мати одне (при тому унікальне) позначення.

Блок-схеми показують весь процес роботи системи з підсистемами та частково доказують правильність вибраних проектних рішень. Тому від точності і детальної блок-схеми залежить результат всієї програми.

При виборі початкової точки відліку при побудові схем було враховано, що виходячи з вибору мови програмування і інших технічних засобів, програма буде об’єктно-орієнтована що вимагає високого рівня декомпозиції задач на класи.

На рисунку 4.1 зображена основна блок-схема програми, на рисунку 4.2 зображено роботу підпрограми.

З яких видно що робота основної програми складається з початкових етапів ініціалізації ПЗ, перевірки наявності ресурсів системи, блоку початку основного циклу з чеканням запиту від користувача в якому відбувається виклик підсистеми та останньої стадії – перевірка поточного стану з завершенням роботи розробленого ПЗ. При роботі підпрограми виконується основний функціонал системи з циклічними послідовностями, перевіркою поточного стану та поверненням в основну програму прапорів стану виконання.

Нижче наведемо ту частину коду, яка виконує вищеперераховані дії.

```
namespace RecoveryDiskCOD
{
    /// <summary>
    /// Клас кодера циклічного коду
    /// </summary>
    public class CicleCodeEncoder : CicleCodeBase
    {
        #region Construction & Destruction
        /// <summary>
        /// Конструктор кодера за замовчуванням
        /// </summary>
        public CicleCodeEncoder()
        {
```

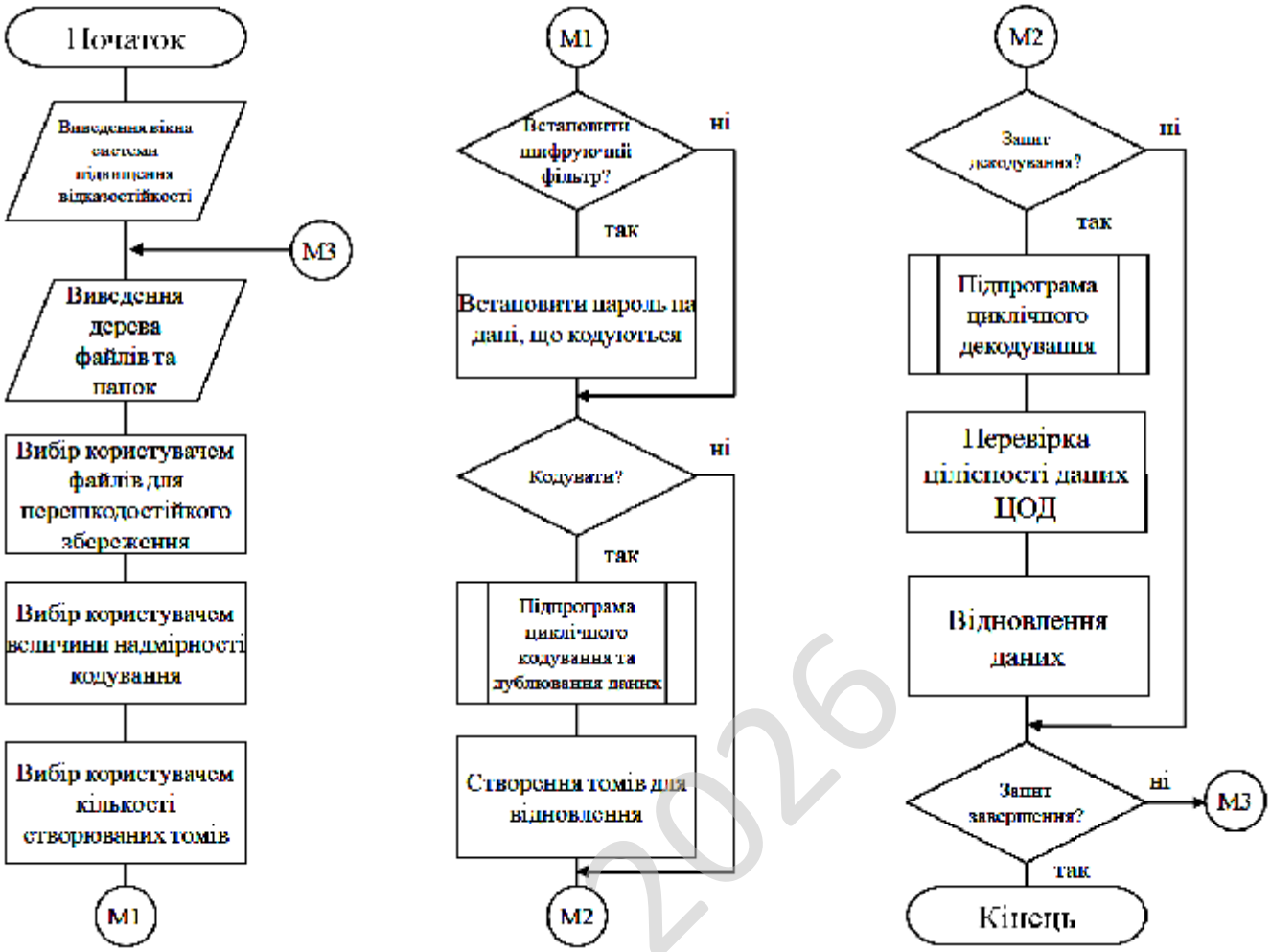

Рисунок 4.1 – Блок-схема основної програми

```
// Створюємо об'єкт класу роботи з елементами поля Галуа
            this.eGF16 = new GF16();
        }
        /// <summary>
/// Базовий конструктор кодера
        /// </summary>
        /// <param name="dataCount">Кількість основних томів</param>
        /// <param name="eccCount">Кількість томів для відновлення</param>
        public CicleCodeEncoder(int dataCount, int eccCount)
        {
// Установка конфігурації кодера
            SetConfig(dataCount, eccCount, (int)CicleCodeType.Alternative);
// Створюємо об'єкт класу роботи з елементами поля Галуа
            this.eGF16 = new GF16();
        }
```

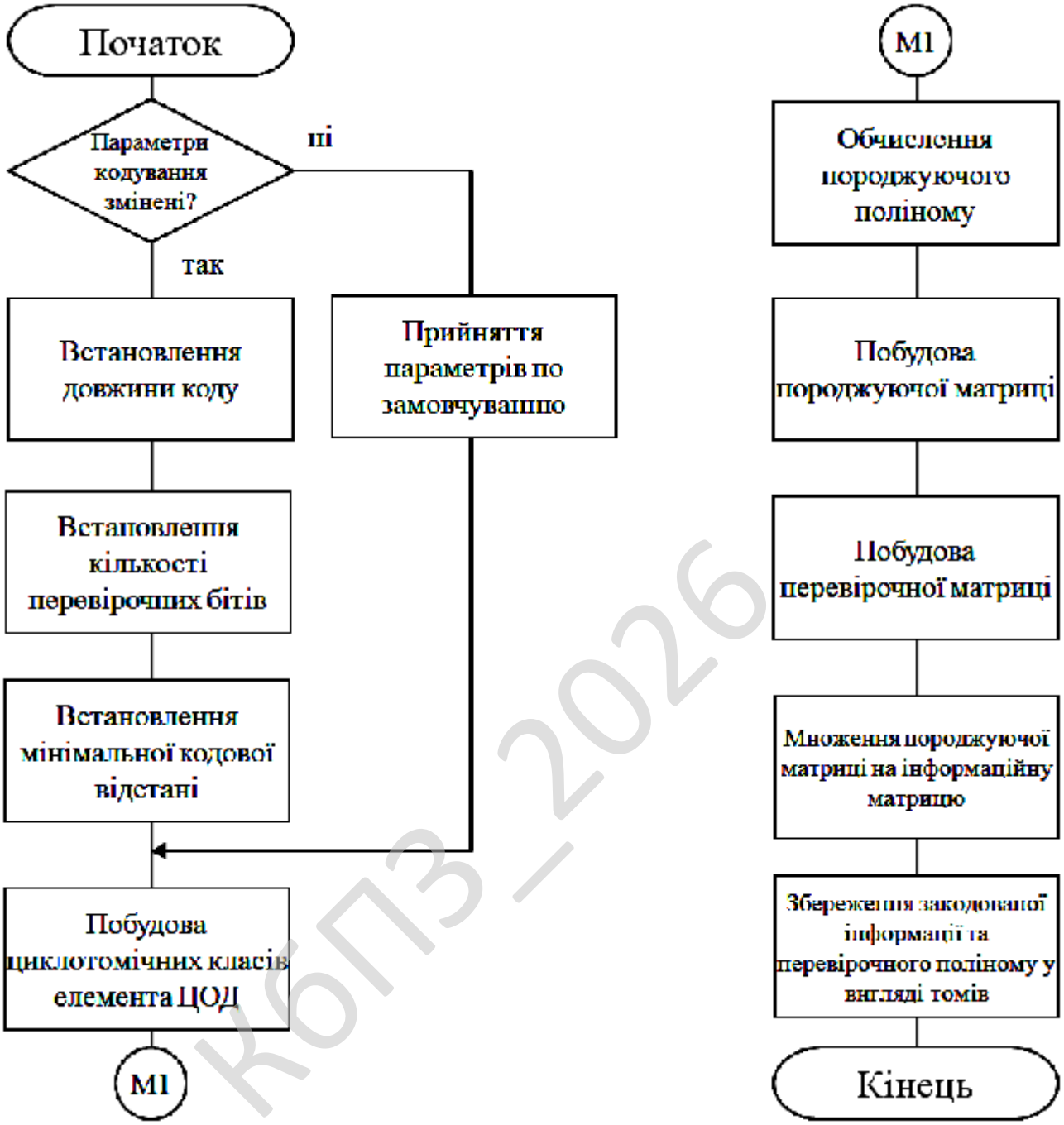

Рисунок 4.2 – Блок-схема роботи підпрограми

```
        /// <summary>
        /// Розширений конструктор кодера
        /// </summary>
        /// <param name="dataCount">Кількість основних томів</param>
        /// <param name="eccCount">Кількість томів для відновлення</param>
        /// <param name="codecType">Тип кодера циклічного коду (по типу
матриці)</param>
```

```
        public CicleCodeEncoder(int dataCount, int eccCount, int codecType)
        {
            // Установка конфігурації кодера
            SetConfig(dataCount, eccCount, codecType);
            // Створюємо об'єкт класу роботи з елементами поля Галуа
            this.eGF16 = new GF16();
        }
        #endregion Construction & Destruction
        #region Public Operations
        /// <summary>
        /// Установка конфігурації кодера
        /// </summary>
        /// <param name="dataCount">Кількість основних томів</param>
        /// <param name="eccCount">Кількість томів для відновлення</param>
        /// <param name="codecType">Тип кодека кодера циклічного коду (по типу
матриці)</param>
        /// <returns>Булевський прапор операції установки конфігурації</returns>
        public bool SetConfig(int dataCount, int eccCount, int codecType)
        {
            int maxVolCount;
            // Установлюємо константи, що відповідають обраному режиму
            if (codecType == (int)CicleCodeType.Dispersal)
            {
                maxVolCount = (int)CicleCodeConst.MaxVolCountDisp;
            } else
            {
                maxVolCount = (int)CicleCodeConst.MaxVolCountAlt;
            }
// Перевіряємо конфігурацію на коректність
            if (
                    (dataCount > 0)
                 &&
                    (eccCount > 0)
                 &&
                    ((dataCount + eccCount) <= maxVolCount)
                )
            {
                // Якщо основна конфігурація змінилася –  сповіщаємо про це
                if (
                        (dataCount != this.n)
                     ||
                        (eccCount != this.m)
```

```
                    ||
                       (codecType != this.eCicleCodeType)
                   )
                {
                    this.mainConfigChanged = true;
                }
                // Зберігаємо конфігурацію
                this.n = dataCount;
                this.m = eccCount;
                this.eCicleCodeType = codecType;
                // Також перераховуємо кількість ітерацій всіх стадій підготовки
                double n = this.n;
                double m = this.m;
// Нормалізуємо значення для розрахунку, щоб уникнути переповнення змінних
                NormalizeNM(ref n, ref m);
// Кількість ітерацій на першій стадії залежить від типу використовуваної матриці
                if (this.eCicleCodeType == (int)CicleCodeType.Alternative)
                {
                    this.iterOfFirstStage = m;
                } else
                {
this.iterOfFirstStage = ((n * m) * n) + (n * ((n + m) + (n * (n + m))));
                }
               this.iterOfSecondStage = 0;
// У кодери немає інвертування матриці

                this.configIsOK = true;
            } else
            {
                //...указуємо на помилку конфігурації
                this.configIsOK = false;
            }
            return this.configIsOK;
        }
        /// <summary>
        /// Метод множення матриці кодування на вхідний прологарифмований вектор
        /// </summary>
        /// <param name="dataLog">Прологарифмований вхідний вектор (вихідні
дані)</param>
        /// <param name="ecc">Вихідний вектор (надлишкові дані)</param>
        /// <returns>Булевський прапор результату операції</returns>
        public bool Process(int[] dataLog, ref int[] ecc)
```

```
        {
// Якщо кодер зконфігуровано некоректно, обробка неможлива!
            if (!this.configIsOK)
            {
                return false;
            }
            // Копіюємо покажчик на масив експонент для скорочення часу обігу
            int[] GF16Exp = this.eGF16.GFExpTable;
            // Обчислення результату множення матриці на вектор
            for (int i = 0; i < this.m; i++)
            {
                int mulSum = 0;         // Сума добутку рядка матриці на стовпець
                int i_n = i * this.n;   // Зсув у масиві до елементів i-ой рядка
                for (int j = 0; j < this.n; j++)
                {
                    mulSum ^= GF16Exp[this.FLog[i_n + j] + dataLog[j]];
                }
                ecc[i] = mulSum;
            }
            return true;
        }
        #endregion Public Operations
        #region Private Operations
        /// <summary>
        /// Заповнення матриці Вандермонда даними
        /// </summary>
        protected override void FillFLog()
        {
// Якщо основна конфігурація змінилася...
            if (this.mainConfigChanged)
            {
                if (this.eCicleCodeType == (int)CicleCodeType.Dispersal)
                {
//...робимо формування дисперсної матриці "D"
                    if (!MakeDispersalMatrix())
                    {
// Указуємо, що кодер зконфігуровано некоректно
                        this.configIsOK = false;
// Активуємо індикатор актуального стану змінних-членів
                        this.finished = true;
// Установлюємо подію завершення обробки
                        this.finishedEvent[0].Set();
```

```
                        return;
                    }
                } else
                {
//...робимо формування альтернативного заповнення матриці "A"
                    if (!MakeAlternativeMatrix())
                    {
                        // Указуємо, що кодер зконфігуровано некоректно
                        this.configIsOK = false;
// Активуємо індикатор актуального стану змінних-членів
                        this.finished = true;
// Установлюємо подію завершення обробки
                        this.finishedEvent[0].Set();
                        return;
                    }
                }
// Виділяємо пам'ять під матрицю "FLog"
                this.FLog = new int[this.m * this.n];
// Заповнюємо матрицю кодування
                for (int i = 0; i < this.m; i++)
                {
                    // Зсув у масиві до елементів i-ой рядка
                    int i_n = i * this.n;
                    // Залежно від типу кодера беремо дані з відповідного масиву
                    if (this.eCicleCodeType == (int)CicleCodeType.Dispersal)
                    {
                        // Формування рядка в матриці кодування
                        for (int j = 0; j < this.n; j++)
                        {
// У матрицю кодування поміщаємо логарифми її вихідних елементів
// (для прискорення множення матриці на вектор)
       this.FLog[i_n + j] = this.eGF16.Log(this.D[((this.n + i) * this.n) + j]);
                        }
                    } else
                    {
                        // Формування рядка в матриці кодування
                        for (int j = 0; j < this.n; j++)
                        {
                            int idx = i_n + j;
// У матрицю кодування поміщаємо логарифми її вихідних елементів
// (для прискорення множення матриці на вектор)
                            this.FLog[idx] = this.eGF16.Log(this.A[idx]);
```

```
                        }
                    }
// У випадку, якщо потрібна постановка на паузу, подію "executeEvent"
// буде скинуто, і будемо на паузі аж до його появи
                    ManualResetEvent.WaitAll(this.executeEvent);
// Якщо зазначено, що потрібно вийти з потоку -  виходимо
                    if (ManualResetEvent.WaitAll(this.exitEvent, 0, false))
                    {
// Указуємо, що кодер зконфігуровано некоректно
                        this.configIsOK = false;
                        // Активуємо індикатор актуального стану змінних-членів
                        this.finished = true;
                        // Установлюємо подію завершення обробки
                        this.finishedEvent[0].Set();
                        return;
                    }
                }

// Якщо є передплата на делегата завершення...
                if (OnCicleCodeMatrixFormingFinish != null)
                {
                    //...повідомляємо, що екземпляр класу готовий до роботи
                    OnCicleCodeMatrixFormingFinish();
                }
                //...і скидаємо прапор
                this.mainConfigChanged = false;
            }
            // Якщо є передплата на делегата завершення...
            if (OnCicleCodeMatrixFormingFinish != null)
            {
                //...повідомляємо, що екземпляр класу готовий до роботи
                OnCicleCodeMatrixFormingFinish();
            }
            // Активуємо індикатор актуального стану змінних-членів
            this.finished = true;
            // Установлюємо подію завершення обробки
            this.finishedEvent[0].Set();
        }
        #endregion Private Operations
    }
}
```

Redmine – вільне серверне ПЗ для управління проектами та відстежування помилок. До системи входить календар-планувальник та діаграми Ганта для візуального представлення ходу робіт за проектом та строків виконання. Redmine написано на мові Ruby і є ПЗ розробленим з використанням відомого веб-фреймворку Ruby on Rails, що означає легкість в розгортанні системи та її адаптації під конкретні вимоги. Для кожного проекту можна вести свої вікі та форуми.

Функціональні можливості:

– Ведення декількох проектів.

– Гнучка система доступу з використанням ролей.

– Система відстеження помилок.

– Діаграми Ганта та календар.

– Ведення новин проекту, документів та управління файлами.

– Сповіщення про зміни за допомогою RSS-потоків та електронної пошти.

– Власна Wiki для кожного проекту.

– Форуми для кожного проекту.

– Облік часових витрат.

– Налаштування власних (custom) полів для задач, затрат часу, проектів та користувачів.

– Легка інтеграція із системами керування версіями (SVN, CVS, Git, Mercurial, Bazaar и Darcs).

– Створення записів про помилки на основі отриманих листів

– Підтримка LDAP автентифікації.

– Можливість самореєстрації нових користувачів.

– Багатомовний інтерфейс (у тому числі українська мова).

– Підтримка СКБД: MySQL, PostgreSQL, SQLite.

Діаграма Ганта (*Gantt chart*, також стрічкова діаграма, графік Ганта) – це популярний тип діаграм, який використовується для ілюстрації плану, графіка

робіт за будь-яким проектом. Є одним з методів планування та управління проектами.

Діаграма Ганта являє собою відрізки (графічні плашки), розміщені на горизонтальній шкалі часу. Кожен відрізок відповідає окремому завданню або підзадачі. Завдання і підзадачі, складові плану, розміщуються по вертикалі. Початок, кінець і довжина відрізка на шкалі часу відповідають початку, кінцю і тривалості завдання. На деяких діаграмах Ганта також показується залежність між завданнями.

Діаграма може використовуватися для представлення поточного стану виконання робіт: частина прямокутника, що відповідає завданню, заштриховується, відзначаючи відсоток виконання завдання; показується вертикальна лінія, що відповідає моменту «сьогодні».

Часто діаграма Ганта використовується спільно з таблицею зі списком робіт, рядки якої відповідають окремо взятій задачі, зображеній на діаграмі, а стовпці містять додаткову інформацію про задачу.

Система відстеження помилок Багтрекер – прикладна програма для допомоги розробникам програмного забезпечення (програмістам, тестувальникам тощо) враховувати і контролювати помилки, знайдені у програмах, питання щодо функціональності, рішення та оновлення, побажання користувачів, а також стежити за процесом їх виконання.

Кожному, хто розробляв програмні продукти, добре знайоме співвідношення «20/80» – останні 20 % роботи тривають 80 % часу.

Як це не парадоксально, але нічого дивного в цій пропорції немає, адже саме на завершальній стадії починається тестування проекту, коли виявляються помилки, і що більший проект, то більше буде знайдено помилок.

Водночас досить часто виявляється, що більшість цих помилок були відомі та могли бути виправлені з меншими витратами на попередніх стадіях роботи, але не були вчасно описані, а потім загубилися серед інших важливих завдань.

Отже, система відстеження помилок у найпростішому варіанті – це процес, що включає в себе виявлення помилки, її опис, виправлення і перевірку цього виправлення, тобто процес «стеження» за багом протягом всього як його життєвого циклу, так і життєвого циклу розробки в цілому.

Сукупність інформації про дефект. Головний компонент такої системи – база даних, що містить відомості про виявлені дефекти. Ці відомості можуть включати в себе:

– номер (ідентифікатор) дефекту;

– хто повідомив про дефект;

– дата і час виявлення дефекту;

– версія продукту, в якій виявлено дефект;

– серйозність (критичність) дефекту та пріоритет рішення;

– опис кроків для відтворення дефекту (неправильної поведінки програми);

– відповідальний за усунення дефекту;

– обговорення можливих рішень та їх наслідків;

– поточний стан виправлення дефекту;

– версії продукту, в якій дефект виправлений.

Крім того, розвинені системи надають можливість прикріплювати файли, які допомагають описати проблему, наприклад, дамп пам'яті або скріншот.

Використання. Основна перевага систем відстеження помилок полягає в забезпеченні чітких централізованих оглядів, запитів на розробку (включаючи помилки і виправлення) та їх стан. У корпоративному середовищі, системи відстеження помилок можуть бути використані для генерації звітів по продуктивності програмістів виправлення помилок. Однак, це може іноді приводити до неточних результатів, тому що різні помилки можуть мати різні ступені пріоритету та серйозності, що пов'язано з складністю їх фіксації.

Життєвий цикл дефекту. Як правило, система відстеження помилок використовує той чи інший варіант «життєвого циклу» помилки, стадія якого визначається поточним станом помилки.

Типовий життєвий цикл дефекту:

1. Новий – дефект зареєстрований тестувальником.

2. Призначений – призначений відповідальний за виправлення дефекту.

3. Дозволений – дефект переходить назад у сферу відповідальності тестувальника. Як правило, супроводжується резолюцією, наприклад:

– Виправлено (виправлення включені у версію таку-то).

– Дубль (повторює дефект, що вже знаходиться в роботі).

– Не виправлено (працює відповідно до специфікації, має занадто низький пріоритет, виправлення відкладено до наступної версії тощо).

– «В мене все працює» (запит додаткової інформації про умови, в яких дефект проявляється).

4. Далі тестувальник проводить перевірку виправлення, залежно від чого дефект або знову переходить у стан «Призначений» (якщо він описаний як виправлений, але не виправлений), або у стан «Закрито».

5. Відкрито повторно – дефект знайдено знову в іншій версії.

Система може надавати адміністраторові можливість налаштування користувачі, які можуть переглядати і редагувати помилки залежно від їх стану, переводити їх в інший стан або видаляти.

У корпоративному середовищі, система відстеження помилок може використовуватися для отримання звітів, що показують продуктивність програмістів при виправленні помилок. Однак, часто такий підхід не дає достатньо точних результатів через те, що різні помилки мають різну ступінь серйозності та складності. При цьому серйозність проблеми прямо не стосується складності її усунення.

Було використано підходи з використанням UML, це уніфікована мова моделювання, використовується у парадигмі об'єктно-орієнтованого програмування. Є невід'ємною частиною уніфікованого процесу розробки програмного забезпечення. UML є мовою широкого профілю, це відкритий стандарт, що використовує графічні позначення для створення абстрактної моделі

системи, називаної UML-моделлю. UML був створений для визначення, візуалізації, проектування й документування в основному програмних систем. UML не є мовою програмування, але в засобах виконання UML-моделей як інтерпретованого коду можлива кодогенерація.

Було використано наступні підходи UML: діаграма діяльності (діаграми поведінки типу); діаграма прецедентів (діаграми поведінки типу).

Діаграма діяльності. Це візуальне представлення графу діяльностей. Граф діяльностей є різновидом графу станів скінченного автомату, вершинами якого є певні дії, а переходи відбуваються по завершенню дій. Дія є фундаментальною одиницею визначення поведінки в специфікації. Дія отримує множину вхідних сигналів, та перетворює їх на множину вихідних сигналів.

Одна із цих множин, або обидві водночас, можуть бути порожніми. Виконання дії відповідає виконанню окремої дії. Подібно до цього, виконання діяльності є виконанням окремої діяльності, буквально, включно із виконанням тих дій, що містяться в діяльності. Кожна дія в діяльності може виконуватись один, два, або більше разів під час одного виконання діяльності. Щонайменше, дії мають отримувати дані, перетворювати їх та тестувати, деякі дії можуть вимагати певної послідовності.

Специфікація діяльності (на вищих рівнях сумісності) може дозволяти виконання декількох (логічних) потоків, та існування механізмів синхронізації для гарантування виконання дій у правильному порядку.

Діаграма прецедентів це діаграма, на якій зображено відношення між акторами та прецедентами в системі. Також, перекладається як діаграма варіантів використання.

Діаграма прецедентів є графом, що складається з множини акторів, прецедентів (варіантів використання) обмежених границею системи (прямокутник), асоціацій між акторами та прецедентами, відношень серед прецедентів, та відношень узагальнення між акторами. Діаграми прецедентів відображають елементи моделі варіантів використання.

Суть даної діаграми полягає в наступному: проектована система представляється у вигляді безлічі сутностей чи акторів, що взаємодіють із системою за допомогою так званих варіантів використання. Варіант використання (use case) використовують для описання послуг, які система надає актору. Іншими словами, кожен варіант використання визначає деякий набір дій, який виконує система при діалозі з актором.

При цьому нічого не говориться про те, яким чином буде реалізована взаємодія акторів із системою.

У мові UML є кілька стандартних видів відношень між акторами і варіантами використання:

– асоціації (association relationship);

– включення (include relationship);

– розширення (extend relationship);

– узагальнення (generalization relationship).

При цьому загальні властивості варіантів використання можуть бути представлені трьома різними способами, а саме – за допомогою відношень включення, розширення і узагальнення.

Відношення асоціації – одне з фундаментальних понять у мові UML і в тій чи іншій мірі використовується при побудові всіх графічних моделей систем у формі канонічних діаграм.

Включення (include) у мові UML – це різновид відношення залежності між базовим варіантом використання і його спеціальним випадком. При цьому відношенням залежності (dependency) є таке відношення між двома елементами моделі, при якому зміна одного елемента (незалежного) приводить до зміни іншого елемента (залежного).

Відношення розширення (extend) визначає взаємозв'язок базового варіанта використання з іншим варіантом використання, функціональна поведінка якого задіюється базовим не завжди, а тільки при виконанні додаткових умов.

4.2 Захист розробленого програмного забезпечення

Розроблене програмне забезпечення захистимо за допомогою наступного алгоритму захисту інформації DSA.

Відправник і одержувач електронного документа використовують при обчисленні великі цілі числа: G і P – прості числа, L біт кожне ($512 < L < 1024$); q – простої число довжиною 160 біт (дільник числа (P-1). Числа G, P, q є відкритими й можуть бути загальними для всіх користувачів мережі.

Відправник вибирає випадкове ціле число X, $1 < X < q$. Число X є секретним ключем відправника для формування електронного цифрового підпису. Потім відправник обчислює значення $Y = G^X \bmod P$. Число Y є відкритим ключем для перевірки підпису відправника. Число Y передається всім одержувачам документів. Цей алгоритм також передбачає використання однобічної функції гешування h(-). У стандарті DSS визначений алгоритм безпечного гешування SHA. Для того щоб підписати документ M, відправник гешує його в ціле геш-значення m: $m = h(M)$, $1 < m < q$, потім генерує випадкове ціле число K, $1 < K < q$, і обчислює число r: $r = (G^K \bmod P) \bmod q$. Потім відправник обчислює за допомогою секретного ключа X ціле число s:

$$s = \frac{m + r * X}{K} \bmod q \ .$$

Пара чисел r і s утворить цифровий підпис $S = (r, s)$ під документом M. Таким чином, підписане повідомлення являє собою трійку чисел $[M,r,s]$. Одержувач підписаного повідомлення $[M, r, s]$ перевіряє виконання умов $0 < r < q$, $0 < s < q$ і відкидає підпис, якщо хоча б одна із цих умов не виконана. Потім одержувач обчислює значення $w = 1/s \bmod q$, геш-значення $m = h(M)$ і числа $u_1 = (m*w) \bmod q$, $u_2 = (r*w) \bmod q$. Далі одержувач за допомогою відкритого ключа Y обчислює значення $v = ((G^{U1} * Y^{U2}) \bmod P) \bmod q$ і перевіряє виконання умови $v = r$. Якщо умова $v = r$ виконується, тоді підпис $S = (r,s)$ під документом M визнається одержувачем справжнім.

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

На рисунку 5.1 зображено розроблене у бакалаврської дипломної роботі програмне забезпечення системи інтелектуального підвищення відказостійкості периферійної ЦОД. З рисунку можна побачити що інтерфейс головного вікна розподілено на наступні функціональні розділи:

– Навігаційного меню яке викликається натисканням правої клавіші маніпулятора миші.

– Верхнього меню: Файл; Інструменти; Параметри; Довідка.

– Функціональних кнопок ПЗ.

– Розділу обрання групи.

– Розділу виведення результату роботи системи.

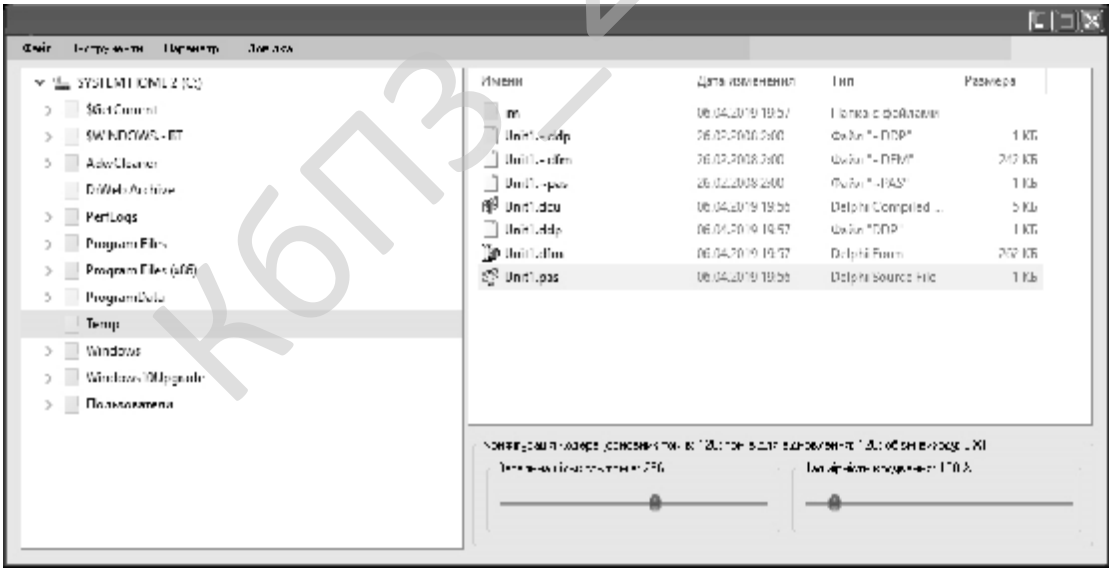

Рисунок 5.1 – Головне вікно ПЗ

Розроблена програма має дуже простий і зрозумілий інтерфейс з користувачем. Кожен, хто в достатньому обсязі володіє операційним

середовищем Windows без особливих складностей освоїть і цю програму, оскільки її інтерфейс інтуїтивно зрозумілий. Якщо програма не видала ніяких помилок, і працює, то можна використовувати, інакше слід слідувати інструкціям, які пропонує програма.

На рисунку 5.2 зображено авторські дані розробленого програмного забезпечення.

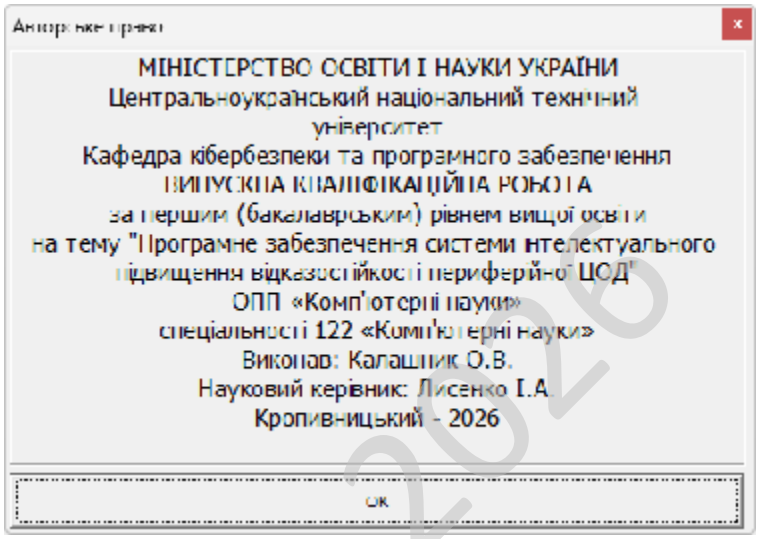

Рисунок 5.2 – Авторське право

Під час роботи над програмою було проведено тестування програмного забезпечення, тобто технічне дослідження, призначене для виявлення інформації про якість продукту відносно контексту, в якому воно має використовуватись.

Тестування включає як процес пошуку помилок або інших дефектів, так і випробування програмних складових з метою їх оцінки.

Проводилась оцінка:

– відповідності поставленим вимогам;

– правильна відповідь для усіх можливих вхідних даних;

– виконання функцій за прийнятний час;

– практичність;

– сумісність з ОС та стороннім ПЗ.

Оскільки число можливих тестів для програмних компонент практично нескінченне, тому стратегія тестування полягала в тому, щоб провести всі можливі тести з урахуванням наявного часу та ресурсів.

Як результат ПЗ тестувалось стандартним виконанням програми з метою виявлення помилок або інших дефектів.

Проводилось тестування чорної скриньки. Основне місце програми тестів «чорної скриньки» – інтерфейс ПЗ. Відомі: функції програми. Досліджується: робота кожної функції на всій області визначення.

Ці тести демонструють:

– Як виконуються функції програми.

– Як приймаються вихідні дані.

– Як виробляються результати.

– Як зберігається цілісність зовнішньої інформації.

При тестуванні «чорної скриньки» розглядаються системні характеристики програм, ігнорується їхня внутрішня логічна структура. Вичерпне тестування, як правило, неможливе. Наприклад, якщо в програмі 10 вхідних величин і кожна приймає по 10 значень, то кількість тестових варіантів становитиме 10^10. Тестування «чорної скриньки» не реагує на багато особливостей програмних помилок. Тестування «чорної скриньки» (функціональне тестування) дозволяє отримати комбінації вхідних даних, які забезпечують повну перевірку всіх функціональних вимог до програми. Програмний виріб тут розглядається як «чорна скринька», чию поведінку можна визначити тільки дослідженням його входів та відповідних виходів. При такому підході бажано мати:

– Набір, утворений такими вхідними даними, які призводять до аномалій у поведінці програми (назвемо його ITc).

– Набір, утворений такими вхідними даними, які демонструють дефекти програми (назвемо його OT).

Будь-який спосіб тестування «чорної скриньки» повинен:

– Виявити такі вхідні дані, які з високою ймовірністю належать набору ITc.

– Сформулювати такі очікувані результати, які з високою імовірністю є елементами набору ОТ.

Принцип «чорної скриньки» не альтернативний принципу «білої скриньки». Скоріше це доповнює підхід, який виявляє інший клас помилок.

Тестування «чорної скриньки» забезпечує пошук наступних категорій помилок:

– Некоректних чи відсутніх функцій.

– Помилок інтерфейсу.

– Помилок у зовнішніх структурах даних або в доступі до зовнішньої бази даних.

– Помилок характеристик (необхідна ємність пам'яті і т.д.).

– Помилок ініціалізації та завершення.

Обрано умови розповсюдження – Freeware. Це власницьке програмне забезпечення, котре можна Безоплатно використовувати протягом необмеженого терміну без обмежень у функціональності, і поширюване без сирцевих кодів. Автори такого програмного забезпечення, як правило, хочуть «дати щось спільноті», але хочуть також контролювати його подальшу розробку. Іноді, коли програмісти вирішують припинити розробку, вони передають сирцевий код іншим програмістам, або ж спільності як вільне програмне забезпечення. Дуже часто плутають поняття «безплатне програмне забезпечення» та «вільне програмне забезпечення», хоча вони суттєво відрізняються. Безплатне програмне забезпечення можна безоплатно встановлювати та використовувати (іноді з певними обмеженнями, як, наприклад, «безплатне для домашнього або некомерційного вжитку»), в той час як вільне програмне забезпечення можна продавати за будь-яку суму, але при тому, у користувача, котрий його отримує, повинні бути права на вивчення, модифікацію та поширення сирцевих кодів одержаної програми.

6 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти, призначено для системи інтелектуального підвищення відказостійкості периферійної ЦОД.

В межах України в недостатній мірі представлені вітчизняні розробки в цій області.

Рішення завдання полягало у вирішенні наступних задач:

– Був проведений огляд існуючих систем інтелектуального підвищення відказостійкості периферійної ЦОД.

– Досліджена система інтелектуального підвищення відказостійкості периферійної ЦОД.

– На основі отриманих результатів досліджень створена програмна реалізація системи інтелектуального підвищення відказостійкості периферійної ЦОД.

Розроблені під час виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти алгоритми дозволяють успішно вирішувати завдання інтелектуального підвищення відказостійкості периферійної ЦОД.

Розроблене програмне забезпечення має простий, дружній та зручний інтерфейс користувача, що забезпечує легкість у освоєнні роботи програмного продукту, зручність у використанні, і не потребує особливих спеціальних знань.

При створені програмного забезпечення було використано об’єктно-орієнтований підхід, що відповідає сучасним тенденціям у галузі розробки комерційних програмних систем.

Програма реалізована на мові високого рівня Visual C#. Дана мова програмування дозволяє найбільш ефективно обробляти дані призначені для системи інтелектуального підвищення відказостійкості периферійної ЦОД. Це

дозволило мінімізувати строк розробки програмного забезпечення, і, як слід, зменшити витрати на його розробку. Запропоноване програмне забезпечення ділиться на загальне програмне забезпечення, що поставляється із засобами обчислювальної техніки й спеціальне програмне забезпечення, що спеціально розроблене для даної конкретної системи й включає програми, що реалізують її функції.

Програма призначена для виконання під управлінням багатозадачної операційної системи Windows 10/11.

Даються необхідні рекомендації з установки розробленого програмного забезпечення.

Для підвищення рівня безпеки запропоновано застосовувати алгоритм DSA.

В цілому створене програмне забезпечення підтверджує правильність використаних проектних рішень та повністю відповідає вимогам технічного завдання. Створене програмне забезпечення має потенційну можливість для подальшого вдосконалення і застосування у різних галузях.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Е. Таненбаум, Д. Уезеролл «Комп'ютерні мережі». – [5-е вид.]. – 2016. – 960 с.

2. Wendell Odom. «CCNA 200-301 Official Cert Guide, Volume 1». Cisco Press. 2020. – 848 p.

3. Wendell Odom. «CCNA 200-301 Official Cert Guide, Volume 2 Premium Edition eBook and Practice Test». Cisco Press. 2020. – 624 p.

4. Scott Jernigan «CompTIA Network+ Certification All-in-One Exam Guide, Eighth Edition». 2022. – 976 p.

5. Doug Lowe «Networking For Dummies 12th Edition». 2020. – 480 p.

6. Ramon Nastase «Computer Networking: The Beginner's guide for Mastering Computer Networking, the Internet and the OSI Model». 2018. – 186 p.

7. Russ White & Ethan Banks «Computer Networking Problems and Solutions: An Innovative Approach to Building Resilient, Modern Networks». 2017. – 832 p.

8. Вінтенко Б., Смірнов О., Миронець І., Смірнова Т., Смірнов С. «Імітаційна модель шляхів вхідних даних комп'ютерної інтелектуальної системи підтримки оператора енергоблоку АЕС». *Комбінаторні конфігурації та їхні застосування: Матеріали XXVII Міжнародного науково-практичного семінару, присвяченого 125-річчю Національного університету «Запорізька політехніка»* (Запоріжжя-Кропивницький-Київ, 4-6 червня 2025 р.). Запоріжжя: НУ «Запорізька політехніка», 2025. С.82-91.

9. Al-Azzeh, J., Ayyoub, B., Mesleh, A., Smirnova, T., Gnatyuk, S., Drieiev, O., Smirnov, O., Dorenskyi, O. «Cloud-Based Information System for Evaluating Caverns in the Process of Blasting Metal Surfaces of Details». *International Review on Modelling and Simulations* 18 (1), 2025. pp. 32-42.

10. Смірнова Т.В., Конопліцька-Слободенюк О.К., Буравченко К.О., Смірнов С.А., Кравчук О.В., Козірова Н.Л., Смірнов О.А. «Дослідження

технологій забезпечення кібербезпеки хмарних сервісів IaaS, PaaS та SaaS». *Кібербезпека: освіта, наука, техніка.* 2024. №4(24), С. 6-27.

11. Батрак О., Смірнова Т., Гнатюк В., Одарченко Р., Смірнов О. «Дослідження показників ефективності функціонування та перспектив розвитку систем IP-телефонії». *Підводні технології,* 2024, № 13, с. 28-35.

12. Kuznetsov, O., Kryvinska, N., Ilchenko, O., Smirnova, T., Ulianovska, Y. «Comparative Analysis of Cryptocurrency Trading Platforms Using the Analytic Hierarchy Process». *CEUR Workshop Proceedings*, 2023, 3628, pp. 106-115.

13. Al-Mudhafar Aqeel, A.M., Smirnova, T., Buravchenko, K., Smirnov, O. «The method of assessing and improving the user experience of subscribers in software-configured networks based on the use of machine learning». *Advanced Information Systems*, 2023, 7(2), pp. 49-56.

14. Smirnov, O., Sydorenko, V., Aleksander, M., Zhyharevych, O., Yenchev, S. «Simulation of the cloud IoT-based monitoring system for critical infrastructures». *CEUR Workshop Proceedings,* Volume 3530, 2023, pp. 256-265.

15. Smirnov, O., Odarchenko, R., Smirnova, T., Bondar, S., Volosheniuk, D. «Optimal Structure Construction of Private 5G Network for the Needs of Enterprises». *Lecture Notes on Data Engineering and Communications Technologies*, 2023, 178, pp. 208–223.

16. Аль-Мудхафар Акіл Абдулхуссейн М., Смірнова Т.В., Буравченко К.О., Смірнов О.А. «Метод оцінки та підвищення користувальницького досвіду абонентів в програмно-конфігурованих мережах на основі використання машинного навчання». *Сучасні інформаційні системи,* 2023, том 7, № 2, С. 49-56.

17. Smirnova, T., Gnatyuk, S., Yudin, O., Sydorenko, V., Polozhentsev, A., «The Model for Calculating the Quantitative Criteria for Assessing the Security Level of Information and Telecommunication Systems». CEUR Workshop Proceedings Volume 3156, 2022, Pages 390-399.

18. Смірнова Т.В., Гнатюк С.О., Сидоренко В.М., Юдін О.Ю., Сидоренко С.Ю., «Модель визначення критичності галузевих інформаційно-

телекомунікаційних систем». *Проблеми інформатизації та управління,* № 2(70). 2022. С. 28-37.

19. Смірнов О.А., Смірнова Т.В., Якименко Н.М., Смірнов С.А., Поліщук Л.І., «Дослідження стійкості до диференціального криптоаналізу запропонованої функції гешування удосконаленого модуля криптографічного захисту в інформаційно-комунікаційних системах» *Системи управління, навігації та зв'язку,* 2022, № 3(69). С. 93-98.

20. Смірнов О.А., Смірнова Т.В., Якименко Н.М., Поліщук Л.І., Смірнов С.А. «Дослідження статистичної стійкості та швидкісних характеристик запропонованої функції гешування удосконаленого модуля криптографічного захисту в інформаційно-комунікаційних системах» *Вісник Хмельницького національного університету. Серія: «Технічні науки»,* № 2 (307). С. 46-52. 2022.

21. Смірнов О.А., Смірнова Т.В., Константинова Л.В., Смірнов С.А., Якименко Н.М., «Дослідження стійкості до лінійного криптоаналізу запропонованої функції гешування удосконаленого модуля криптографічного захисту в інформаційно-комунікаційних системах» *Системи управління, навігації та зв'язку*, 2022, № 1(67). С. 84-89.

22. Smirnova T., Gnatyuk S., Berdibayev R., Avkurova Zh., Iavich M. «Cloud-Based Cyber Incidents Response System and Software Tools». *Communications in Computer and Information Science*, 2021, vol 1486. Springer, Cham. pp 169-184.

23. Smirnov O., Kuznetsov A., Kiian A., Kuznetsova T. «Non-binary constant weight coding technique». *CEUR Workshop Proceedings*. Volume 2740, 2020, Pages 102-114.

24. Smirnov O., Alimseitova Zh., Adranova A., Akhmetov B., Lakhno V., Zhilkishbayeva G. «Models and algorithms for ensuring functional stability and cybersecurity of virtual cloud resources». *Journal of theoretical and applied information technology* Vol.98. No 21, 2020, P. 3334-3346.

25. Smirnov O., Kuznetsov A., Kiian A., Cherep A., Kanabekova M., Chepurko I. «Testing of code-based pseudorandom number generators for post-quantum application». *2020 IEEE 11th International Conference on Dependable Systems, Services and Technologies (DESSERT),* Ukraine, Kyiv, May 14-18. 2020. P. 172-177.

26. Smirnov O., Kuznetsov A., Pushkar'ov A., Serhiienko R., Babenko V., Kuznetsova T., «Representation of Cascade Codes in the Frequency Domain». In: Radivilova T., Ageyev D., Kryvinska N. (eds) Data-Centric Business and Applications. Lecture Notes on Data Engineering and Communications Technologies, vol 48. Springer, Cham. 2021. pp 557-587.

27. Smirnov, O., Markovets, O. Vovk, N., Turchyn, Y., «Model of informational support for social network administrators' content creation». *CEUR Workshop Proceedings* Volume 2616, 2020, Pages 125-136.

28. Smirnov, O., Drieieva, H., Drieiev, O., Polishchuk, Y., Brzhanov, R., Aleksander, M. «Method of fractal traffic generation by a model of generator on the graph». *CEUR Workshop Proceedings* Volume 2616, 2020, Pages 366-379.

29. Smirnov, O., Drieieva, H., Drieiev, O., Simakhin, V., Bondar, S., Odarchenko, R. «Managing multifractal properties of the binary sequence generated with the Markov chains», *CEUR Workshop Proceedings* Volume 2608, 2020, Pages 633-645.

30. Smirnov O. Kuznetsov A., Zaicenko Yu., Pastukhov M., Oleshko O., Kuznetsova K., «Formation of Discrete Signals with Special Correlation Properties». *International Conference on Information and Telecommunication Technologies and Radio Electronics, UkrMiCo 2019*; Odessa; Ukraine; 9-13 September 2019. P.22-28.

31. Smirnov, O., Kuznetsov, A., Kolovanova, I., Kuznetsova, T., «Noise immunity of the algebraic geometric codes». International Journal of Computing; 2019, Volume 18, Issue 4 – Research Institute for Intelligent Computer Systems – 2019. – P. 393-407.

32. Smirnov, O., Kuznetsov, A., Reshetniak, O., Ivko, N., Katkova, T., Kuznetsova, T., «Generators of Pseudorandom Sequence with Multilevel Function of Correlation». *2019 IEEE International Scientific-Practical Conference Problems of Infocommunications, Science and Technology (PIC S&T),* Kyiv, Ukraine, 8 – 11 October 2019 . P.517-522.

33. Smirnov, O., Odarchenko, R., Abakumova, A., Usik, P., Kundyz, M., «QoE optimization technique for media delivery in 5G networks». *2019 IEEE International Scientific-Practical Conference Problems of Infocommunications, Science and Technology (PIC S&T),* Kyiv, Ukraine, 8 – 11 October 2019. P.597-601.

34. Smirnov, O., Krasnobayev, V., Yanko, A., Kuznetsova, T. «Methods of nulling numbers in the system of residual classes». *CEUR Workshop Proceedings*, Vol 2588, P. 90-106, 2019.

35. Smirnov, O., Kuznetsov, A., Kovalchuk, D., Averchev, A., Pastukhov, M., Kuznetsova, K., «Formation of Pseudorandom Sequences with Special Correlation Properties», *2019 3rd International Conference on Advanced Information and Communications Technologies, AICT -2019*/ Lviv, Ukraine, 2-6 July, 2019, P. 395-399.

36. Smirnov, O., Kuznetsov, A., Kiian, A., Zamula, A., Rudenko, S., Hryhorenko, V., «Variance Analysis of Networks Traffic for Intrusion Detection in Smart Grids», *2019 IEEE 6th International Conference On Energy Smart Systems (2019 IEEE ESS)*, Kyiv, Ukraine April 17-19, 2019 P. 353-358.

37. Smirnov, O., Kuznetsov, A., Kavun, S., Babenko, B., Nakisko, O., Kuznetsova, K., «Malware Correlation Monitoring in Computer Networks of Promising Smart Grids», *2019 IEEE 6th International Conference On Energy Smart Systems (2019 IEEE ESS)*, Kyiv, Ukraine April 17-19, 2019 P. 347-352.

38. Smirnov, O., Kuznetsov, A., Kovalchuk, D., Pastukhov, M., Kuznetsova, K., Prokopovych-Tkachenko, D., «Discrete Signals with Special Correlation Properties», *CEUR Workshop Proceedings* Volume 2353, *CEUR Workshop Proceedings* 2019, Pages 618-629.

39. Smirnov A.A., Kuzne tsov A.A., Danilenko D.A., Berezovsky A., «The statistical analysis of a network traffic for the intrusion detection and prevention systems», *Telecommunications and Radio Engineering*. – Volume 74, Issue 1. – Begel House Inc. – 2015. – P. 61-78.

40. Смірнов О.А., Смірнова Т.В., Буравченко К.О., Кравченко С.С., Горбов В.О., «Хмарна система підтримки прийняття рішень технологічного процесу відновлення поверхонь конструкцій і деталей машин». *Сучасні інформаційні системи*. 2021. Т. 5, № 4. С. 79-95

41. Смірнов О.А., Усік П.С., Миронець І.В., Буравченко К.О., Якименко Н.М. «Метод підвищення ефективності розподіленої обробки даних у комп'ютерних системах операторів стільникового зв'язку» *Вісник Черкаського державного технологічного університету. Технічні науки.* №4. С. 103-110. 2020.

42. О.А.Смірнов, Т.В.Смірнова, Л.І. Поліщук, К.О. Буравченко, А.О.Макевнін, «Дослідження хмарних технологій як сервісів», *Кібербезпека: освіта, наука, техніка*. № 3(7). С. 43-62. 2020.

43. Смірнов О.А., Конопліцька-Слободенюк О.К., Смірнов С.А., Буравченко К.О., Смірнова Т.В., Поліщук Л.І. Інформаційна безпека в комп'ютерних мережах. Навчальний посібник – Кропивницький: вид. Лисенко В.Ф. 2020. – 294 с.

44. О.А. Смірнов, П.С. Усік, «Дослідження перспектив використання технологічних рішень в мережах 5G» у *Кібербезпека та інформаційні технології: монографія.* – Х. : ТОВ «ДІСА ПЛЮС», 2020.С. 122-135.

45. Смірнов О.А., Дрєєва Г.М., Дрєєв О.М., Смірнова Т.В. «Фрактальний аналіз генератора самоподібного трафіку на основі ланцюга Маркова». *Центральноукраїнський науковий вісник. Технічні науки.* № 2(33). с. 161-172, 2019.

46. Смірнов О.А., Конопліцька-Слободенюк О.К., Смірнов С.А., Буравченко К.О., Смірнова Т.В. Поліщук Л.І. Проектування комп'ютерних

систем та мереж. Навчальний посібник – Кропивницький: вид. Лисенко В.Ф. 2019. – 264 с.

47. Smirnov, O., Kuznetsov, A., Kuznetsova., K. Synthesis of Discrete Signals with Improved Correlation Properties. Монографія: In.: ISCI'2019: Information Security in Critical Infrastructures. Collective monograph. Edited by Ivan D. Gorbenko and Alexandr A. Kuznetsov, ASC Academic Publishing, USA, 2019, pp. 281-299. – ISBN: 978-0-9989826-8-7 (Hardback), ISBN: 978-0-9989826-9-4 (Ebook).

48. Смірнов О.А., Дрєєва Г.М. Метод генерування фрактального трафіку за допомогою моделі генератора на графі. Монографія: Інформаційна безпека та інформаційні технології : монографія / за заг. ред. В. С. Пономаренка. – Х. : Вид. Рожко С.Г. 2019. С. 123-139

49. Дрєєва Г.М., Смірнов О.А., Дрєєв О.М. Метод генерування фрактальноподібної числової послідовності на основі скінченного автомату для моделювання трафіку у мережі. Центральноукраїнський науковий вісник. Технічні науки. № 1(32). с. 173-183, 2019.

50. Смірнова Т.В., Солових Є.К., Смірнов О.А., Дрєєв О.М. Побудова хмарних інформаційних технологій оптимізації технологічного процесу відновлення та зміцнення поверхонь деталей. Центральноукраїнський науковий вісник. Технічні науки. № 1(32). с. 184-194, 2019.

51. Смірнов О.А., Смірнов С.А., Поліщук Л.І., Смірнова Т.В., Конопліцька-Слободенюк О.К. Метод формування антивірусного захисту даних з використанням безпечної маршрутизації метаданих. Кібербезпека: освіта, наука, техніка. – Том 3 № 3. – Київ: КУ ім. Бориса Грінченка. – 2019. – С. 63-87.

52. Смірнов О.А., Гнатюк С.О., Кавун С.В., Терейковський І.А., Жмурко Т.О., Смірнов С.А., Коваленко А.С. Основи безпеки в комп'ютерних мережах. Навчальний посібник – Кропивницький: вид. Лисенко В.Ф. 2018. – 177 с.