Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет комп'ютерних наук та технологій

Кафедра комп'ютеризованих систем управління

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри

_____Олександр ЛИТВИНЕНКО

«____» _____2023 р.

КВАЛІФІКАЦІЙНА РОБОТА
(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ЗДОБУВАЧА ВИЩОЇ ОСВІТИ
ОСВІТНЬОГО СТУПЕНЯ «МАГІСТР»

Тема: Програмний модуль планування технічного обслуговування повітряних суден

Виконавець: _____Олександр МАЛІЦЬКИЙ

Керівник: _____Олександр ЛИТВИНЕНКО

Нормоконтролер: _____Євгеній ТУПОТА

Київ 2023

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет комп’ютерних наук та технологій
Кафедра комп’ютеризованих систем управління
Спеціальність 123 «Комп'ютерна інженерія»
(шифр, найменування)

Освітньо-професійна програма «Системне програмування»
Форма навчання денна

ЗАТВЕРДЖУЮ
Завідувач кафедри

_____Олександр ЛИТВИНЕНКО

«____»_____ 2023 р.

ЗАВДАННЯ
на виконання кваліфікаційної роботи
Маліцького Олександра Олексійовича
(прізвище, ім'я, по батькові випускника в родовому відмінку)

1. Тема кваліфікаційної роботи(проєкту): «Програмний модуль планування технічного обслуговування повітряних суден» затверджена наказом ректора від «28» серпня 2023р. №1497/ст.

2. Термін виконання роботи (проєкту): з 02.10.2023 р. по 24.12.2023р.

3. Вихідні дані до роботи (проєкту): Технологія планування технічного обслуговування повітряних суден.

4. Зміст пояснювальної записки:

1. Аналітичний огляд літературних джерел з тематики кваліфікаційної роботи.
2. Дослідження евристичного методу розв’язання задач складання розкладу виконання робіт.
3. Програмна реалізація модулю планування технічного обслуговування повітряних суден.

5. Перелік обов'язкового графічного (ілюстративного) матеріалу:

1. Планування задач (схема алгоритму);
2. Форма інтерфейсу користувача;
3. Діаграма класів системи.

6. Календарний план-графік

№ пор.	Завдання	Термін виконання	Відмітка про виконання
1	Дослідити технологію організації авіаційних перевезень	02.10.2023 – 05.10.2023	
2	Проаналізувати існуючі методи маршрутизації авіаційних перевезень	06.10.2023 – 13.10.2023	
3	Дослідити евристичні методи побудови маршрутів	14.10.2023 – 02.11.2023	
4	Адаптувати евристичний алгоритм комівояжера до розв'язання задачі маршрутизації авіаційних перевезень	02.11.2023 – 28.11.2023	
5	Визначити технічне та інформаційне забезпечення програмного модуля маршрутизації авіаційних перевезень	29.12.2023 – 01.12.2023	
6	Здійснити програмну реалізацію евристичного алгоритму побудови маршрутів авіаційних перевезень	02.12.2023 – 10.12.2023	
7	Розробити інтерфейс програмного модуля маршрутизації авіаційних перевезень	11.12.2023 – 14.12.2023	
8	Скласти та надати керівнику пояснювальну записку кваліфікаційної роботи у повному обсязі, а також текст та презентацію доповіді	17.12.2023 – 18.12.2023	
9	Підготовка матеріалів для передачі секретарю ДЕК (ПЗ, *CD-R* з електронними копіями ПЗ, презентації, відгук керівника, рецензія)	19.12.2023 – 20.12.2023	

7. Дата видачі завдання: «2» жовтня 2023 р.

Керівник кваліфікаційної роботи _____ Олександр ЛИТВИНЕНКО
(підпис керівника) (П.І.Б.)

Завдання прийняв до виконання _____ Олександр МАЛІЦЬКИЙ
(підпи студента) (П.І.Б.)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи «Програмний модуль планування технічного обслуговування повітряних суден»: 80 сторінок, 8 рисунків, 3 таблиці, 33 використаних джерел.

ТЕХНІЧНЕ ОБСЛУГОВУВАННЯ, ПОВІТРЯНЕ СУДНО, ПЛАНУВАННЯ, ЕВРИСТИЧНИЙ АЛГОРИТМ, ПЕРЕСТАНОВКА.

Об’єктом кваліфікаційної роботи є процес технічного обслуговування повітряних суден.

Предметом дослідження є евристичний метод планування технічного обслуговування повітряних суден.

Методами дослідження є сучасні методи розробки, мова програмування *C#*, фреймворк *.NET*.

Метою кваліфікаційної роботи є розробка програмного модулю планування технічного обслуговування повітряних суден на основі евристичного методу.

Результатом виконання кваліфікаційної роботи є розроблений застосунок для планування задач технічного обслуговування. Новизна отриманих результатів полягає в застосуванні евристичного алгоритму в контексті планування технічного обслуговування повітряних суден.

Технологічними характеристиками можна виділити здатність алгоритму адаптуватися до непередбачуваних умов, швидкість реагування на зміни, можливість інтеграції з іншими системами управління обслуговування або базами даних.

Рекомендації щодо використання результатів кваліфікаційної роботи полягають в можливій подальшій спробі впровадження алгоритму в сфері технічного обслуговування повітряних суден для авіаційних компаній або обслуговуючих підприємств.

Зміст

Вступ

Актуальність теми. Сучасний стан планування технічного обслуговування піддається критиці через ряд факторів, серед яких можна виділити відсутність інтегрованих систем планування, відсутність чітких графіків планування обслуговування суден, та неоптимальні стратегії обслуговування. Багато авіакомпаній використовують розділені системи для моніторингу та планування технічного обслуговування. Відсутність інтегрованих рішень може ускладнювати оптимальне розподіл ресурсів. Більшість систем планування реагує на виявлені проблеми, а не передбачає їх, прогнозування технічних витрат та ресурсів може сприяти більш ефективному використанню часу та коштів. Через можливість використання неоптимальних стратегій обслуговування, стратегії вибору часу та місця технічного обслуговування, можливе виникнення сиутації що може впливати на ефективність використання повітряних суден.

У контексті постійних змін та вірогідності непередбачуваних ситуацій в авіаційній галузі важливо мати інструмент, який дозволяє швидко обробити необхідну інформацію, щоб зробити практичні рішення. Такий розроблюваний модуль може виступати важливим ресурсом для авіакомпаній, ремонтних бригад, сервісів технічного обслуговування повітряних суден.

Мета і завдання виконання кваліфікаційної роботи. Мета виконання кваліфікаційної роботи – розробка програмного модулю для планування технічного обслуговування повітряних суден, який відповідає сучасним вимогам, надає користувачам зручний та ефективний інструмент для реаліації або планування робіт в галузі ремонтних робіт авіації.

Технічне обслуговування повітряних суден є ключовим елементом забезпечення їхньої безперервної та безпечної експлуатації, в свою чергу евристичні алгоритми можуть відігравати важливу роль у плануванні цих обслуговувань, допомагаючи забезпечити оптимальні рішення шляхом поетапного вибору найбільш перспективних варіантів. Їх застосування дозволяє враховувати технічний стан судна, інтенсивність його експлуатації та інші фактори, що сприяє ефективному та раціональному плануванню технічного обслуговування повітряних суден.

Завдання роботи включають:

1. Дослідити особливості технічного обслуговування повітряних суден;
2. Аналіз існуючих типів та засобів вирішення задач;
3. Вивчити та дослідити роботу планування задач;
4. Розробити зручний і швидкодіючий програмний модуль.

Мета кваліфікаційної роботи – створити інноваційний програмний модуль, який забезпечить користувачам зручний та ефективний спосіб регулювання планування технічного обслуговування повітряних суден.

Об’єкт і предмет дослідження. Об’єкт кваліфікаційної роботи – створення програмного модулю планування технічного обслуговування повітряних суден.

Предмет кваліфікаційної роботи – модуль планування технічного обслуговування повітряних суден.

Методи дослідження. Щоб досягти виконання зазначених завдань кваліфікаційної роботи було вирішено застосувати навички по роботі в інтегрованому середовищі розробки *Visual Studio*. Дане середовище було обрано через його здатність об'єднувати процеси розробки та підтримувати різні технології.

Мовою програмування для реалізації алгоритмів було обрано *C#.* Мова була вибрана через велику кількість підтримуваних технологій та бібліотек, а саме чудову взаємодію фреймворком *.NET,* який дозволяє створювати системи будь-яких типів.

Наукова новизна отриманих результатів. Новизна отриманих результатів полягає в застосуванні евристичного алгоритму в контексті планування технічного обслуговування повітряних суден. Такий підхід дозволяє забезпечити оптимальне використання ресурсів та підвищити загальну ефективність планування та проведення обслуговування літаків на відміну від існуючих методик.

Практичне значення отриманих результатів. Результатом виконання кваліфікаційної роботи є розроблений програмний модуль планування технічного обслуговування повітряних суден. Згідно з отриманими даними, результати

кваліфікаційної роботи свідчать про можливість застосуваня алгоритму для планування та регулювання технічного обслуговування повітряних суден.

Особистий внесок випускника. Всі результати, представлені у кваліфікаційній роботі отримані випускником особисто. Особливої уваги заслуговує проєктування структури модулю планування з використанням евристичного алгоритму важливого завдання першим.

Прогнозні припущення про розвиток об'єкту та предмету дослідження. Припущення полягають у можливості застосування розробленого програмного модулю як системи для планування графіків обслуговування повітряних суден. Подальший розвиток даного модулю відкриває багато перспектив та можливостей для покращення сфери авіації.

РОЗДІЛ 1
АНАЛІЗ СТАНУ ПРОБЛЕМИ ТА ПОСТАНОВКА ЗАДАЧІ ДОСЛІДЖЕННЯ

1.1. Технологія технічного обслуговування повітряних суден

Технологія технічного обслуговування повітряних суден відіграє вирішальну роль у забезпеченні безпеки та надійності експлуатації повітряних суден. Зі зростанням складності сучасних авіаційних систем стає обов'язковим мати передові технології технічного обслуговування, щоб відповідати вимогам авіаційної промисловості.

Технологія технічного обслуговування повітряних суден охоплює широкий спектр заходів, включаючи перевірку, усунення несправностей, ремонт і профілактичне обслуговування. Це передбачає використання різних інструментів, обладнання та процедур для забезпечення оптимального робочого стану літака. Метою технічного обслуговування повітряних суден є підтримання максимальної продуктивності літака, мінімізація часу простою та забезпечення безпеки пасажирів і екіпажу. Щоб краще зрозуміти особливості види технічного обслуговування та їх значення для забезпечення якісної та ефективної роботи в різних частинах обслуговування було зображено у таблиці 1.1.

Таблиця 1.1

Види технічного обслуговування

Вид	**Опис**	**Значення**

Структурне обслуговування	Регулярні перевірки та ремонт структур літака для забезпечення безпеки та довговічності.	Забезпечення стійкості та міцності конструкції, попередження виникнення дефектів.
Двигун та система потужності	Періодичні перевірки, технічне обслуговування і ремонт авіамоторів	Забезпечення надійності і ефективності двигуна, попередження можливих поломок.
Системи електроніки та навігації	Обслуговування та оновлення бортових систем	Забезпечення актуальності та ефективності систем управління та навігації.
Системи безпеки та аварійного реагування	Перевірка та технічне обслуговування систем, які забезпечують безпеку в разі аварій та непередбачених ситуацій.	Гарантування швидкого та ефективного реагування на аварійні ситуації.
Гідромеханічні системи	Обслуговування гідроциліндрів, гідравлічних систем та інших механічних компонентів.	Забезпечення працездатності та надійності гідравлічних систем.
Паливна система	Перевірка та технічне обслуговування системи постачання пального для забезпечення надійності та ефективності.	Гарантування правильної подачі пального для оптимальної роботи двигуна.
Системи теплозабезпечення та кондиціонування	Обслуговування систем, які забезпечують комфорт та безпеку для екіпажу та пасажирів.	Забезпечення приємних умов для роботи та подорожування.
Регулярні огляди та технічні перевірки	Проведення регулярних технічних оглядів	Виявлення потенційних проблем та їх розв'язання до виникнення серйозних поломок.
Оновлення та модернізація	Здійснення оновлень та модернізацій для вдосконалення безпеки та відповідності стандартам.	Підтримка сучасних технічних стандартів та підвищення загальної продуктивності повітряного судна.

Проаналізувавши таблицю 1.1, можна зробити висновки щодо видів обслуговування.

Структурне обслуговування визначається як ключовий етап, спрямований на запобігання дефектів та забезпечення стійкості конструкції. Обслуговування

двигунів та системи потужності має велике значення для гарантування безперебійної роботи та ефективності повітряного судна. Системи електроніки та навігації вимагають регулярного обслуговування для забезпечення точності та безпеки в польотах. Системи безпеки та аварійного реагування визначаються як критичні для швидкого та ефективного реагування на непередбачені ситуації. Гідромеханічні системи, паливні системи та системи теплозабезпечення вносять суттєвий внесок у функціональність та комфорт повітряних суден. Регулярні технічні огляди та перевірки є необхідним елементом для виявлення та усунення потенційних проблем. Оновлення та модернізація є стратегічними для відповідності сучасним технічним стандартам та підвищення продуктивності літака.

Одним із ключових досягнень у технології технічного обслуговування повітряних суден є використання комп’ютеризованих систем управління технічним обслуговуванням і систем моніторингу стану літаків. Ці системи збирають дані в режимі реального часу про характеристики різних систем і компонентів літака. Аналізуючи ці дані фахівці з технічного обслуговування можуть виявити потенційні проблеми та вирішувати їх завчасно, зменшуючи ризик збоїв у польоті.

CMMS (Computerized Maintenance Management System) – це комп'ютеризована система управління технічним обслуговуванням, яка використовується для планування, виконання та контролю технічного обслуговування та ремонту обладнання та активів в організації. *CMMS* дозволяє підприємствам ефективно управляти технічним обслуговуванням, підтримувати оптимальний стан обладнання і знижувати витрати на ремонт та обслуговування.

CMMS також оптимізує процедури технічного обслуговування шляхом автоматизації трудомістких завдань, таких як планування перевірок, керування записами технічного обслуговування та відстеження запасів запасних частин. Це спрощує процес обслуговування, підвищує ефективність і знижує витрати.

Іншим ключовим аспектом технології обслуговування літаків є розробка передових діагностичних інструментів і обладнання. Розвиток технологій технічного обслуговування літаків призвів до використання більш надійних і довговічних матеріалів у конструкції літаків.

Наприклад, використання композитних матеріалів у конструкціях літаків зменшило потребу в частих перевірках і ремонтах, традиційно пов’язаних з металевими конструкціями. Це призводить до зниження витрат на технічне обслуговування та підвищення доступності літака.

Традиційні методи технічного обслуговування повітряних суден в основному покладалися на підходи до технічного обслуговування за розкладом і за часом. Ці методи передбачають проведення планових перевірок, переважно на основі заздалегідь визначених інтервалів часу, годин польоту або циклів.

Ці передбачувані графіки технічного обслуговування допомагають виявляти й усувати потенційні проблеми за допомогою профілактичних дій. Крім того, використовуються методи реактивного технічного обслуговування, щоб реагувати на конкретні несправності або збої, виявлені під час перевірок або про які повідомляють пілоти.

Технологія технічного обслуговування повітряних суден базується на використанні спеціалізованого обладнання та процедур для забезпечення безпеки, надійності та ефективності повітряних суден. Основні технології технічного обслуговування повітряних суден включають в себе:

Щодо ПС цивільної авіації радянського та пострадянського виробництва встановлено такі види ТО: поточне (або лінійне), періодичне, сезонне, спеціальне, та при зберіганні.

Кожен з видів ТО відрізняється обсягом і складністю робіт, тривалістю і періодичністю їх виконання. Поточне (лінійне) ТО виконується безпосередньо перед вильотом і після посадки ПС в базових, транзитних і кінцевих аеропортах. При цьому виконуються наступні види робіт: по зустрічі ПС, забезпечення стоянки, огляду і обслуговування, забезпечення вильоту.

Основне призначення поточного (лінійного) технічного обслуговування – усунення відмов і пошкоджень, що виникли у польоті, і підготовка повітряних суден до чергового вильоту. При поточному ТО, як правило, не повинно бути робіт,

необхідність виконання яких визначається нальотом (числом посадок) ПС або індивідуально напрацюванням його окремих агрегатів і виробів.

Необхідність, частота і послідовність виконання оперативних форм ТО обумовлюється характером і умовами використання ПС за призначенням. Роботи щодо забезпечення вильоту виробляються безпосередньо перед вильотом ПС незалежно від того, яка форма оперативного ТО виконувалася.

Періодичне технічне обслуговування виконується через певний період часу, що вимірюються числом годин нальоту ПС, числом посадок або календарним часом. Основне призначення періодичного ТО – виявлення і усунення відмов і пошкоджень елементів, виробів і агрегатів функціональних систем ПС на ранніх стадіях їх розвитку, а також проведення профілактичних заходів щодо запобігання виникненню відмов і пошкоджень при подальшій його експлуатації: заміна агрегатів, які відпрацювали ресурс, мастило шарнірних з'єднань, регулювання виробів за результатами технічного діагностування і інші заходи, спрямовані на підтримку працездатності здатності систем. Виконання періодичних форм ТО забезпечує підтримку працездатності та необхідної справності парку ПС.

Форми періодичного технологічного обслуговування відрізняються значно більшою трудомісткістю і чіткою періодичністю виконання. Процесом технічного обслуговування повітряних суден зазвичай керують: Авіакомпанії – надають послуги літакам якими оперують; ОАМ (виробник обладнання) – сертифіковані виробники, які виконують технічне обслуговування своєї продукції; Сертифіковані компанії з надання послуг технічного обслуговування.

Саме планування технічного обслуговування повітряних суден зазвичай здійснюється в рамках технічного відділу, або служби авіакомпанії, або організації. Цей відділ або служба відповідають за розробку і виконання графіку технічного обслуговування і ремонту літаків. Вони розробляють графік обслуговування, планують і визначають перелік необхідних робіт, виконують моніторинг стану повітряних суден, відстежують терміни і відповідності до стандартів безпеки.

Підсумовуючи інформацію про існуючі методи технічного обслгуовування суден можна зробити висновок, що однією з ключових переваг евристичного методу

технічного обслуговування є його здатність адаптуватися до індивідуальної поведінки літака та експлуатаційних варіацій. На відміну від традиційних методів, евристичне технічне обслуговування враховує конкретне використання та стан літака, що дозволяє складати точніші та персоналізовані плани техобслуговування. Метод використовує дані в реальному часі та алгоритми машинного навчання для виявлення закономірностей і тенденцій, що дозволяє обслуговуючому персоналу приймати обґрунтовані рішення щодо втручання та оптимізації.

1.2. Аналіз методів планування технічного обслуговування

Методи планування технічного обслуговування (ТО) повітряних суден грають важливу роль у забезпеченні їх безпеки, надійності та ефективності. Розглянемо методи технічного обслуговування та проведемо аналіз їх ефективності.

Базове обслуговування – ключовий етап в технічному обслуговуванні повітряних суден. Цей процес проводиться періодично, зазвичай кожні 1 – 2 роки, і забезпечує ретельне технічне оглядовування та ефективний ремонт повітряного судна. Основні складові цього етапу включають:

1. Розбирання частин повітряного судна. Проводиться демонтаж та розбирання ключових частин літака для доступу до важливих компонентів;
2. Огляд та заміна компонентів. Кожна частина повітряного судна проходить ретельний технічний огляд. Пошкоджені або зношені компоненти замінюються новими для забезпечення безпеки та надійності;
3. Відновлення структурних елементів. Проводиться перевірка структурних елементів на знос та можливі пошкодження. В разі необхідності здійснюється їх відновлення чи заміна;
4. Системний аналіз. Проводиться комплексний аналіз роботи систем повітряного судна для виявлення та усунення можливих несправностей.

Періодичні огляди і ремонти – систематичні перевірки та ремонти, що виконуються відповідно до графіка, визначеного виробником або регулюючими органами.

Детальний опис включає:

1. Перевірка структурних компонентів. Виконується детальна перевірка стану структурних елементів повітряного судна з метою виявлення будь-яких слабких місць чи пошкоджень;
2. Заміна систем. Системи повітряного судна перевіряються на працездатність, і в разі виявлення несправностей або застарілості може здійснюватися їх заміна;
3. Тестування безпеки. Проводиться комплексне тестування систем безпеки для забезпечення їхньої ефективності та відповідності стандартам;
4. Оновлення обладнання. Проводяться оновлення обладнання та програмного забезпечення для відповідності останнім стандартам та покращення функціональності.

Калібрування і тестування обладнання – важлива частина забезпечення точності та ефективності систем повітряного судна. Вона включає:

1. Періодичне калібрування. Бортові прилади, системи навігації, радіообладнання та інше обладнання піддаються регулярному калібруванні для забезпечення їхньої точності;
2. Тестування бортових систем. Здійснюється комплексне тестування бортових систем для визначення їх працездатності та можливих проблем.
3. Проводження тестів, що імітують аварійні ситуації, допомагає переконатися в тому, що системи повітряного судна можуть ефективно впоратися з непередбачуваними обставинами та забезпечити безпеку пасажирів і екіпажу.

Заміна та модернізація обладнання передбачає впровадження новітніх технологій та покращень для підвищення продуктивності та безпеки повітряного судна:

1. Підвищення продуктивності. Заміна застарілого обладнання новітніми системами дозволяє підвищити ефективність та знизити ризик несправностей;
2. Покращення безпеки. Встановлення сучасних систем безпеки та навігації допомагає підвищити рівень безпеки повітряного судна;
3. Відповідність стандартам. Модернізація здійснюється з урахуванням останніх стандартів безпеки і навігації.

Кожен з цих етапів є необхідним для забезпечення безпеки, надійності та ефективності повітряних суден, враховуючи строгі стандарти цивільної авіації та вимоги виробників.

Технічне обслуговування повітряних суден - це критичний аспект цивільної авіації, оскільки воно забезпечує безпеку та надійність польотів та пасажирських перевезень. Методи планування технічного обслуговування (ТО) повітряних суден мають свої переваги та недоліки, які важливо враховувати при виборі стратегії обслуговування. Вище наведена таблиця включає в себе різні методи та їх характеристики. Щоб порівняти переваги та недоліки між існуючими методами обслуговування їх було зображено у таблиці 1.2.

Таблиця 1.2

Переваги та недоліки існуючих методик

Метод	Переваги	Недоліки
Регулярний план технічного обслуговування	Вчасне виконання обслуговування, зменшення ризику аварій.	Надмірне або недостатнє обслуговування, збільшення часу простою обладнання.
Прогнозування на основі стану обладнання	Максимальне використання ресурсів,	Витрати на діагностику, недооцінка стану обладнання.
Запланована заміна обладнання	Уникнення витрат на невідкладні ремонти, збільшення доступності обладнання.	Збільшення витрат на запасні частини, непередбачені витрати на плановану заміну.
Оптимізація вартості	Ефективне використання ресурсів.	Суб'єктивність вибору методу, може призводити до недостатнього рівню обслуговування в деяких випадках.
Проактивна стратегія технічного обслуговування	Уникнення несподіваних поломок, збільшення надійності та тривалості обладнання.	Вимагає аналізу та прогнозу стану обладнання,можливе збільшення витрат на діагностику та обслуговування.
Гнучка стратегія технічного обслуговування	Адаптація до змін у виробничому середовищі, можливість швидкого реагування на потреби.	Вимагає великої гнучкості та планування, ризик неефективного використання ресурсів.

Проаналізувавши інформацію, слід зазначити що базове обслуговування є ключовим етапом, оскільки воно включає регулярне технічне оглядовування та ремонт повітряного судна. Перевагою цього методу є те, що він дозволяє виявляти та усувати проблеми на ранніх стадіях, забезпечуючи безпеку та надійність. Однак його недолік полягає в тому, що вимагає значних витрат часу та ресурсів.

Періодичні огляди і ремонти, хоча і виконуються за графіком, також важливі для забезпечення довговічності та ефективності повітряних суден. Вони дозволяють проводити систематичні перевірки та заміни компонентів, підтримуючи високий рівень безпеки.

Калібрування та тестування обладнання є необхідним етапом для забезпечення точності та ефективності бортових систем. Це дозволяє уникнути ситуацій, пов'язаних із відхиленням від стандартів та недоліками в роботі обладнання.

Заміна та модернізація обладнання вносять важливий внесок у підвищення продуктивності та безпеки повітряних суден. Встановлення новітніх технологій та систем безпеки допомагає враховувати сучасні стандарти та підтримувати конкурентоспроможність на ринку.

У висновках можна підкреслити, що комплексне використання усіх зазначених методів технічного обслуговування є необхідним для досягнення комплексної безпеки, надійності та ефективності повітряних суден. Застосування цих методів дозволяє враховувати вимоги сучасних стандартів цивільної авіації та забезпечує конкурентоспроможність авіакомпаній на ринку повітряного транспорту.

Після врахування цих плюсів та мінусів важливо здійснювати баланс та вибирати методи обслуговування, які найкращим чином відповідають конкретним умовам та вимогам експлуатації повітряних суден. Згідно з аналізом гнучкі та інтегровані стратегії планування ТО можуть бути ефективним рішенням для досягнення оптимального балансу між ефективністю та витратами.

1.3. Постановка задачі дослідження

Важливо розуміти, що кожен інструмент для вирішення дослідження має свої переваги та недоліки, і вибір залежить від конкретних потреб користувача. Все це свідчить про важливість інформаційних технологій у сучасному світі та їхню роль у розвитку авіаційної галузі.

Метою даного дослідження є розробка програмного модуля, який надасть ефективний інструмент для планування технічного обслуговування повітряних суден

з урахуванням їхньої експлуатаційної історії, технічного стану та інших ключових параметрів. Серед основних задач для дослідження можна виділити:

1) Визначення вимог до програмного модуля.

Мета: Визначення конкретних потреб і очікувань від програмного модуля планування технічного обслуговування.

Завдання: Провести опитування експертів, збір вимог від фахівців та операторів повітряних суден, аналіз сучасних стандартів та вимог до безпеки авіаційних операцій.

2) Розробка алгоритмів та структури модуля.

Мета: Створення ефективних алгоритмів для визначення оптимальних інтервалів технічного обслуговування та розробка інтерфейсу програмного модуля.

Завдання: Розробка математичних моделей, імплементація алгоритмів планування.

3) Імплементація та тестування модуля.

Мета: Створення функціонального програмного продукту та перевірка його ефективності та стабільності.

Завдання: Програмування програмного модуля, проведення різноманітних тестів, аналіз результатів і виправлення виявлених помилок.

4) Оцінка ефективності та порівняння з існуючими методами.

Мета: Визначення переваг та обмежень розробленого модуля в порівнянні з існуючими методами технічного обслуговування.

Завдання: Проведення експериментів з реальними даними, аналіз результатів, порівняння з ефективністю традиційних методів.

Очікується, що результатом дослідження буде розроблений та протестований програмний модуль, який враховує необхідність використання новітніх технологій у технічному обслуговуванні повітряних суден.

Цей модуль:

1. Забезпечить точніше визначення оптимальних інтервалів технічного обслуговування на основі аналізу технічного стану та історії експлуатації повітряних суден;
2. Гарантуватиме стабільну роботу модуля навіть в умовах змінних технічних параметрів сучасних повітряних суден та різних сценаріїв експлуатації;
3. Призведе до зменшення витрат на технічне обслуговування та підвищення безпеки авіаційних операцій завдяки більш точному та адаптивному плануванню.

1.4. Висновки до розділу

У даному розділі ми детально розглянули технологію технічного обслуговування повітряних суден. Цей аналітичний огляд виявив особливості та переваги різних видів облуговування.

У сучасному авіаційному технічному обслуговуванні важливе значення приділяється впровадженню комп'ютеризованих систем управління. Ці системи дозволяють збирати та аналізувати дані в режимі реального часу, що сприяє виявленню потенційних проблем і прийняттю негайних заходів для їх усунення. Замість традиційного підходу, що базується на планових перевірках, комп'ютеризовані системи надають можливість більш оперативного та точного контролю за станом повітряних суден.

Було проаналіовано спроможність евристичного технічного обслуговування на відміну від традиційних методів адаптуватися до індивідуальних особливостей літака та експлуатаційних варіацій дозволяє враховувати конкретні умови експлуатації, що призводить до точніших та персоналізованих стратегій технічного обслуговування.

Технічне обслуговування повітряних суден є складним та відповідальним процесом, спрямованим на забезпечення їх безпеки, надійності та ефективності. Основні етапи технічного обслуговування можна розглядати як стратегічний підхід до догляду за літаками, що включає базове обслуговування, періодичні огляди та

ремонти, калібрування та тестування обладнання, а також заміну та модернізацію обладнання. Кожен з цих етапів виконує визначену роль у забезпеченні нормативів та вимог цивільної авіації, гарантуючи високий ступінь безпеки та ефективності в експлуатації повітряних суден.

Основні завдання дослідження включають визначення вимог до програмного модуля, розробку ефективних алгоритмів та структури, оцінку його ефективності порівняно з існуючими методами технічного обслуговування.

Звідси випливає, що результатом дослідження очікується функціональний програмний модуль, який сприятиме точнішому визначенню оптимальних інтервалів технічного обслуговування, забезпечить стабільну роботу навіть в умовах змінних технічних параметрів та призведе до зменшення витрат на технічне обслуговування та підвищення безпеки авіаційних операцій завдяки більш точному та адаптивному плануванню.

РОЗДІЛ 2
ЕВРИСТИЧНИЙ МЕТОД ПЛАНУВАННЯ ТЕХНІЧНОГО ОБСЛУГОВУВАННЯ ПОВІТРЯНИХ СУДЕН

2.1. Евристичний метод складання розкладів виконання робіт

Алгоритми стали важливим інструментом в області інформаційних технологій, забезпечуючи ефективне та швидке розв'язання певних класів задач. Вони використовуються у різноманітних сферах, включаючи оптимізацію, обробку графів, розподілені системи та інші області. Однією з основних областей використання алгоритмів є задачі оптимізації. Вони успішно вирішують проблеми мінімізації або максимізації функціоналів, знаходячи локально оптимальні рішення. Прикладом може слугувати алгоритм вибору мінімального шляху для ефективної доставки товарів. У графових задачах алгоритми використовуються для знаходження оптимальних шляхів, найменших остовних дерев та інших структур.

Серед основних видів алгоритмів для вирішення задачі складання розкладів робіт можна виділити два основних види – жадібні та евристичні.

Жадібні алгоритми можуть бути ефективними в розподілених системах, де важливо приймати локально оптимальні рішення для забезпечення ефективного використання ресурсів та мінімізації часу виконання завдань. Вони дозволяють швидко знаходити локально оптимальні рішення, хоча при цьому не гарантується завжди глобальна оптимальність. Враховуючи їхню простоту та швидкодію, жадібні алгоритми залишаються важливим інструментом для інженерів і розробників в області інформаційних технологій.

Хоча жадібні алгоритми та евристичні алгоритми можуть використовуватися для оптимізації, жадібні алгоритми є більш обмеженим підходом. Вони часто вибирають локально оптимальні рішення, сподіваючись на глобальну оптимальність, тоді як евристичні алгоритми більш гнучкі та можуть використовувати різні стратегії, щоб знаходити приблизні оптимальні рішення в більш різноманітних умовах.

Евристичні методи можуть швидко надавати прийнятні рішення без великих обчислювальних витрат, що робить їх ефективними в умовах обмежених термінів. Евристичні задачі розкладу можуть бути використані при вхідних нечітких або

невизначених параметрів або даних. Саме через швидкість та варіативність евристичні методи можуть ефективно працювати в умовах обмежених або мінімальних даних, дозволяючи швидко знаходити рішення.

Евристичний метод є підходом до розв'язання проблем, який базується на емпіричних правилах, досвіді та інтуїції, а не на строгих математичних моделях. У випадку оптимізації розкладу задач, евристика використовується для знаходження прийнятного рішення в області задачі планування та графіків.

Щоб проаналізувати та обрати найкращий метод наступним кроком буде необідно перерахувати всі можливі алгоритми які можуть бути використані для побудови розкладу виконання робіт, та обрати найкращий.

Алгоритм найкоротшого часу виконання першим (*SPT – Shortest Processing Time*)

Алгоритм є ефективним у випадках, коли відома невелика кількість завдань і користувачеві важливо мінімізувати час завершення всіх завдань. Проте не завжди результат алгоритму є оптимальним і часто може призводити до довших середніх часів очікування для деяких завдань.

Важливо врахувати, що випадку великої кількості завдань метод *SPT* може не бути оптимальним, оскільки йому може бракувати стратегічного планування.
Метод не завжди ефективний у ситуаціях, де потрібна гнучкість та здатність до адаптації до змін. Метод не враховує важливості завдань або стратегічних пріоритетів, що може бути недоцільним для деяких сценаріїв.

Найважчі завдання першими (*Most Work Remaining First*).

Цей метод визначає пріоритет завдань на основі обсягу залишкової роботи, що залишилася до їх завершення. Завдання з більшою кількістю залишкової роботи отримують вищий пріоритет, оскільки їх завершення може вимагати більше уваги та

зусиль. Це підходить для ситуацій, де важливо оптимізувати використання ресурсів та забезпечити ефективне вирішення завдань з великим обсягом роботи.

Слід зазначити, що метод не завди може бути найоптимальнішим в деяких ситуаціях. Завдання з більшою кількістю залишкової роботи можуть мати тенденцію затримувати завдання з коротшим терміном виконання, що може викликати проблеми з дотриманням графіка та термінів. Сам метод може не враховувати важливості завдань, приділяючи пріоритет завданням із значущим обсягом роботи, але меншою стратегічною вагою. Метод може сприяти завданням з великим обсягом роботи, незалежно від їхньої важливості, що може викликати нерівномірність у вирішенні менших, але вірогідно важливих завдань.

Метод призначення ресурсів (*Greedy Assignment of Resources*).

Цей метод вирішує завдання розподілу ресурсів на основі їхнього впливу на критерії оптимізації. Завдання з більшим впливом на досягнення цілей отримують перевагу при призначенні ресурсів. Такий підхід допомагає максимізувати використання ключових ресурсів та сприяє досягненню стратегічних цілей, визначаючи їхню важливість для загального успіху проєкту чи бізнесу.

Метод також не є ідеальним, адже зосередження на впливових завданнях може призвести до нерівномірного розподілу ресурсів та недостатньої уваги до інших, менш впливових, але важливих завдань, що може порушити баланс та стабільність системи. Вірогідні можливі затримки для завдань. Завданням із середнім впливом може бути призначено менше ресурсів, що може викликати затримки в їх виконанні, навіть якщо вони мають стратегічне значення. Впливу кожного завдання може вимагати значних обчислювальних ресурсів, особливо при великій кількості завдань, що може вплинути на продуктивність.

Переваги: метод дозволяє ефективно розподіляти та використовувати ресурси для досягнення максимально оптимального виконання завдань. Цей підхід гнучко адаптується до змін у вимогах та умовах проекту, що дозволяє ефективно управляти ресурсами протягом усього проекту.

Недоліки: вірогідний ризик можливих затримок через неоптимальний розподіл ресурсів, а також недооцінка важливості деяких завдань, що може вплинути на кінцеві результати проекту.

Метод найменшого використання ресурсів (*Least Slack Time*).

Цей метод враховує залишковий час між поточним часом і дедлайном завдань. Завдання з меншим залишковим часом отримують вищий пріоритет, оскільки їхнє швидке виконання дозволяє ефективно реагувати на зміни. Підхід найменшого використання ресурсів сприяє оперативному призначенню важливих завдань, забезпечуючи своєчасне та ефективне виконання.

Метод ймовірно може ігнорувати стратегічне значення завдань та їх важливість для загальної мети проєкту, оскільки основний акцент робиться на терміновості.

Моливе застосування тільки залишкового часу може призвести до недооцінки важливості завдань, що може вплинути на стратегічне планування. В певних сценаріях, наприклад, коли деякі завдання мають однаковий термін виконання, метод може бути менш ефективним у визначенні пріоритетів.

Переваги: метод спрямований на економне використання ресурсів та мінімізацію витрат. Це може бути особливо важливим у ситуаціях з обмеженими ресурсами, де ефективне розподілення обов'язків може зменшити загальні витрати.

Недоліки: метод може призвести до простоїв через недостатню увагу до ресурсів та їхнього потенціалу. Ігнорування важливості деяких завдань може також вплинути на успішність проекту в довгостроковій перспективі.

Найменший дедлайн першим (*Earliest Due Date - EDD*).

Цей метод призначає пріоритет завдань на основі їхніх крайніх термінів виконання. Завдання з найближчим дедлайном отримують вищий пріоритет, наділяючи їх важливістю у контексті часових обмежень. Основна ідея полягає в

тому, щоб вирішувати спочатку ті завдання, які мають найменший час до дедлайну. мінуси

Метод зосереджений лише на часових обмеженнях, ігноруючи важливість та стратегічне значення завдань. Застосування алгоритму може призвести до перевантаження, оскільки завдання з найближчим дедлайном можуть вимагати більше уваги, і це може бути неефективним у довгостроковій перспективі. Також метод може бути менш гнучким у вирішенні змінюваних умов або пріоритетів, оскільки акцент робиться на жорстких часових рамках. Оскільки метод базується лише на крайніх термінах, його прозорість і передбачуваність можуть бути обмеженими, що ускладнює адаптацію до змін. Важливо також те, що метод ігнорує можливі зв'язки та залежності між завданнями, що може впливати на загальну ефективність планування.

Переваги: метод покликаний максимізувати швидкість виконання завдань та прискорювати процес проекту. Це особливо ефективно в умовах, де час є критичним фактором.

Мінуси: вірогідний шанс виникнення ризику конфліктів між ресурсами у змагальних умовах, що може призвести до невірного розподілу завдань та втрати ефективності.

Найбільш важливе завданням першим (*Weighted Shortest Job First*).

Цей евристичний метод не обмежується тільки тривалістю завдань, але також враховує їхню важливість. Отримуючи вищий пріоритет, завдання з більшою важливістю або меншою тривалістю, метод створює баланс між часом виконання та стратегічним значенням.

Це особливо корисно, коли деякі завдання мають визначальне стратегічне значення, і їхнє швидке виконання стає критичним для успіху проєкту чи задачі. Враховуючи своїм особливостям метод сприяє оптимальному розподілу ресурсів і ефективному,але не завжди вірному вирішенню ключових завдань.

Цей метод особливо корисний в умовах, коли деякі завдання мають стратегічне значення, і їхнє вчасне виконання є вирішальним для успішного завершення проекту чи виконання задачі.

Враховуючи це, *Weighted Shortest Job First* створює баланс між ефективністю виконання та стратегічним значенням, забезпечуючи, що ключові завдання отримують високий пріоритет.

Сам евристичний є потужним інструментом для планування завдань, оскільки враховує не тільки їхню тривалість, але й важливість. Прикладом може слугувати планування важливого інноваційного проекту в сфері технологій.

При використанні цього методу, менеджер проекту визначає, які завдання мають визначальне стратегічне значення для успіху проекту.

Наприклад, розробка ключового модулю, який визначає конкурентоспроможність продукту. Це завдання може мати вагомий вплив на успіх впровадження проекту та взаємодію з клієнтами.

У цьому контексті, *Weighted Shortest Job First,* алгоритм присвоює важливому завданню вищий пріоритет, навіть якщо воно не є найдовшим за тривалістю. Це дозволяє забезпечити, що ключові аспекти проекту будуть виконані швидко і ефективно, забезпечуючи високий стартовий рівень успішності та задоволення від клієнтів.

Система може бути налаштована на динамічне оновлення пріоритетів у реальному часі, враховуючи нові умови, вимоги та стратегічні зміни. Це дозволяє більш гнучко реагувати на динаміку проекту.

При використанні цього алгоритму управління проектом може виявити ключові аспекти, які допоможуть у підвищенні конкурентоспроможності продукту чи сервісу. Враховуючи стратегічне значення, менеджер проекту може доручити важливі завдання командам, які можуть ефективно і швидко вирішувати їхні вимоги.

У практичних реалізаціях метод важливого завдання першим є важливим для:

1. Визначення критеріїв (мінімізація часу виконання, максимізація використання ресурсів);
2. Початкового розташування (вибір початкового стану за допомогою емпіричних правил);

3. Послідовного додавання завдань до розкладу на основі їхнього впливу на критерії оптимізації. Завдання можуть бути вибрані за допомогою евристичних правил, таких як призначення найбільш важливих або термінових задач;
4. Періодичного оновлення розкладу шляхом видалення, перегляду або перерозподілу завдань, які не відповідають поточним вимогам чи умовам;
5. Оцінки якості поточного розкладу та внесення коректив на основі отриманих результатів.

Сама ідея алгоритму полягає в тому, щоб обирати завдання для виконання відповідно його значення для виконання першим (часу обробки).

При використанні методу розкладання можна спробувати вперше призначати завдання, які мають найменший термін виконання або найбільший вплив на критерії оптимізації.

Однак важливо бути готовим до корекцій та оновлень розкладу для досягнення кращих результатів в процесі його реалізації.

Отримання оптимального розкладу завдань у програмі має значуще практичне значення у різних областях діяльності. Завдяки цьому можна досягти покращення ефективності та управління ресурсами.

Система розкладу дозволяє оптимізувати використання робочого часу, зменшуючи час виконання завдань та мінімізуючи час очікування.

Всі ці фактори в свою чергу призводять до збільшення продуктивності, якості обслуговування та економії ресурсів.

Оптимізація розкладу корисна в контексті управління проєктами, виробництва, обслуговування клієнтів та багатьох інших галузях. Важливою перевагою є можливість адаптації до змінних обставин та змінюваних умов, що робить програму гнучкою та здатною до швидкого втручання в реальному часі.

Загалом, розклад завдань сприяє підвищенню ефективності процесів, управлінню ресурсами та забезпеченню вчасного та ефективного виконання завдань, що веде до загального покращення продуктивності.

Евристичні методи дозволяють здійснити розв'язання складних задач, для яких людина не може дати точний (математичний) алгоритм. Евристичні методи відтворюють логіку міркувань людини в процесі розв'язання конкретної задачі.

Однак, на відміну від традиційних методів, які здійснюють виведення рішень на основі моделей представлення знань про предметну область, евристичні методи не вимагають наявності таких моделей, а логіка міркувань людини відбувається (моделюється) самим алгоритмом.

У найпростішій класичній постановці задача побудови розкладів формулюється наступним чином.

Є множина однакових верстатів:

$$V = (v_j \mid j = \overline{1,n}), \tag{2.1}$$

де v_j – позначення j-го верстату;

n – кількість верстатів.

Задана множина (список) робіт, які необхідно виконати за допомогою цих верстатів:

$$R = (r_i \mid i = \overline{1,m}), \tag{2.2}$$

де r_i – позначення i -ї роботи;

m – кількість робіт.

Відома тривалість τ_i виконання кожної i -ї роботи:

$$T = (\tau_i \mid i = \overline{1, m}). \quad (2.3)$$

Необхідно так розподілити роботи між верстатами і упорядкувати їх у часі, щоб загальна тривалість їх виконання була мінімальною. Верстати можуть працювати одночасно. Будь-яку роботу можна виконувати на будь-якому верстаті. Процес виконання роботи верстатом не переривається до його завершення.

Шуканий розклад виконання робіт описується вектором значень бівалентних змінних:

$$X = (x_{yk} \mid i = \overline{1, m}; j = \overline{1, n}; 1 \leq k \leq m - n). \quad (2.4)$$

Зміст формули полягає у наступному: якщо $x_{ijk} = 1$, це означає, що i -а робота виконується на j -у верстаті k -ю за рахунком; при $x_{ijk} = 0$ це твердження невірне.

Евристичні алгоритми побудови розкладів виконання робіт ґрунтуються на таких правилах:

1) До виконання приймаються роботи у порядку не зростання тривалості їх виконання;

2) На початковому етапі верстати завантажуються роботами у довільному порядку (наприклад, у порядку їх нумерації);

3) На подальших етапах верстати завантажуються роботами у порядку їх звільнення від виконання попередніх робіт.

Найпростіший евристичний алгоритм побудови розкладу виконання робіт передбачає послідовну m -кратну реалізацію однотипних обчислювальних циклів – окремо для кожної роботи, тобто $r_i \in R$.

Нехай R^* – множина робіт, які на початок поточного циклу ще не розглядалися; $R^* \subseteq R$;

$$I^* \subseteq \{i = \overline{1,m}\} \tag{2.5}$$

де I^* – множина номерів таких робіт;

k_j^* – номер останньої позиції у розкладі j -го верстата, для якої робота вже визначена.

Обчислювальний цикл складається з таких дій:

1) Із множини R^* робіт, які ще не були розглянуті, вибирається робота r_{i^*} з найбільшою тривалістю виконання:

$$\tau_i = \{\tau_i \mid i \in I^*\}. \tag{2.6}$$

2) Обчислюються моменти часу t^B_{j,k_j^*} звільнення верстатів від виконання попередніх робіт:

$$t^B_{j,k_j^*} = t^K_{j,k_j^*}, \tag{2.7}$$

де t^K_{j,k_j^*} – час завершення виконання попередньої роботи, яка займає k_j^*-у позицію у розкладі j -го верстата;

$j = \overline{1,n}$;

$t^K_{j,0} = 0$.

3) Обирається верстат для виконання роботи r_j.

Для виконання роботи r_j обирається верстат v_j, який раніше інших звільняється від виконання попередньої k_j^*-ї за рахунком роботи:

$$t^B_{j^*,k_{j^*}^*} = \{t^B_{j,k_{j^*}^*} \mid j = \overline{1,n}\}. \tag{2.8}$$

4) Змінній $x_{i^*,j^*,k_{j^*}^*+1}$ привласнюється значення$x_{i^*,j^*,k_{j^*}^*+1} = 1$.

Це означає, що робота r_{i^*} буде виконуватися верстатом v_{j^*} $(k_{j^*}^* + 1)$ -ю по черзі.

5) Із множини R^* видаляється елемент r_{i^*}, а із множини I^* – елемент i^*.

6) Обчислюється час завершення виконання роботи r_{i^*} верстатом v_{j^*}:

$$t_{j^*,k_{j^*}^*+1}^K = t_{j^*,k_{j^*}^*}^K + \tau_{i^*}. \tag{2.9}$$

Обчислювальний цикл реалізується m разів, поки не будуть розподілені між верстатами і упорядковані у часі всі роботи, що входять до множини R.

Після цього визначається загальна тривалість процесу виконання всього комплексу робіт:

$$T^{\Sigma} = \{\sum_{i \in I_j} \tau_i \mid j = \overline{1,n}\}, \tag{2.10}$$

де I_j – множина номерів робіт, які заплановані до виконання на j -у верстаті.

$$I_j = \{i: \ (1 \le i \le m)\&(1 \le k \le m - n)\&(x_{ijk} = 1)\}, \tag{2.11}$$

де $j = \overline{1,n}$.

Послідовність операцій алгоритму зображено на схемі послідовності дій алгоритму побудови розкладів, зображеній на рис. 2.1.

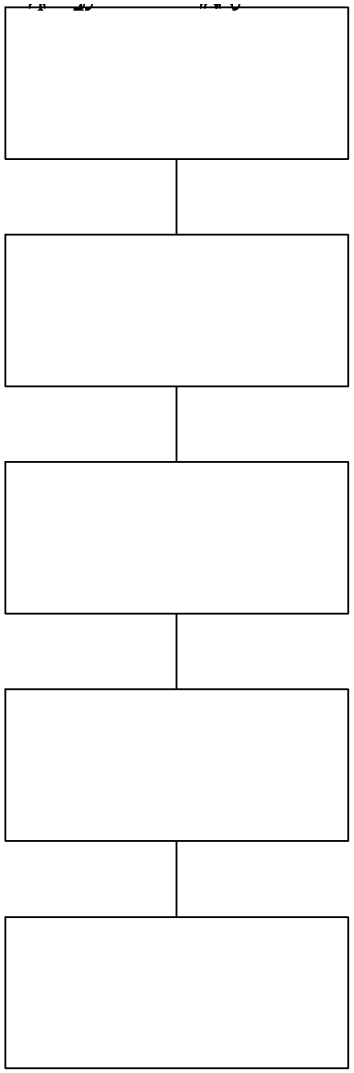

Рис. 2.1. Послідовність дій при побудові розкладів

2.2. Застосування евристичного методу складання розкладу виконання робіт до планування технічного обслуговування повітряних суден

Комп’ютеризована система управління технічним обслуговуванням визначається як ключовий елемент технології планування технічного обслуговування. Ця інтегрована система надає авіакомпаніям та технічним службам можливість ефективно контролювати та керувати всіма аспектами обслуговування літаків. Система забезпечує зручний механізм відстеження та планування усіх видів робіт, управління робочими замовленнями, ефективне управління запасами та ведення детальної історії технічного обслуговування.

Це технологічне рішення дозволяє авіакомпаніям оптимізувати процеси обслуговування літаків, зменшити можливість помилок та мінімізувати час простою літаків. Вона також сприяє забезпеченню надійності та безпеки польотів через систематичне та контрольоване проведення технічних робіт. Комп'ютеризована система управління технічним обслуговуванням є важливим інструментом для оптимізації та підвищення ефективності процесів в авіаційній та технічній галузях.

Система надає централізовану базу даних, яка дозволяє в режимі реального часу бачити стан кожного літака та вимоги до його технічного обслуговування.

Також важливою технологія, яка використовується для планування технічного обслуговування повітряних суден являється програмне забезпечення для прогнозного технічного обслуговування. Це програмне забезпечення використовує різні датчики та системи моніторингу для збору даних про компоненти та системи літака в реальному часі. Постійно аналізуючи ці дані, програмне забезпечення для прогнозованого обслуговування може виявити потенційні проблеми до того, як вони стануть критичними. Це дозволяє бригадам з технічного обслуговування завчасно планувати необхідні завдання з технічного обслуговування, запобігаючи незапланованим простоям і знижуючи ймовірність збоїв або аварій під час польоту.

Ці системи призначені для моніторингу роботи критичних систем і компонентів під час польоту. Моніторинг у режимі реального часу допомагає виявити будь-які аномалії або потенційні збої, дозволяючи своєчасно планувати технічне обслуговування та підвищувати загальну безпеку. Технічне обслуговування повітряних суден є невід'ємною частиною їх експлуатації. Воно включає в себе періодичне перевірку стану повітряних суден та виконання запланованих робіт з технічного обслуговування, що необхідні для забезпечення їх безпеки та ефективності.

Існують різні методи планування технічного обслуговування повітряних суден, які дозволяють ефективно виконувати цю задачу. Один із методів планування технічного обслуговування повітряних суден – це метод визначення залишкового ресурсу.

Залишковий ресурс встановлюється на підставі аналізу витрат на обслуговування певного агрегату повітряного судна. Завдяки цьому аналізу, можна визначити момент, коли агрегат або система потребуватимуть ремонту або заміни. Такий метод дозволяє максимально ефективно використовувати ресурси та уникнути непланових збоїв у роботі повітряного судна.

Іншою методикою планування технічного обслуговування повітряних суден являється метод МВТ. Метод відновлювального технічного обладнання передбачає розбиття обладнання, яке слід обслуговувати, на окремі деталі та системи. Кожній деталі присвоюється власне значення, щодо його впливу на діяльність повітряного судна.

На підставі цього методу розробляється графік планування технічного обслуговування, який враховує всі деталі та системи повітряного судна, а також їхні інтервали обслуговування. Цей метод сприяє вчасному виявленню та усуненню можливих проблем з технічним обладнанням повітряного судна.

Основна ідея цього методу полягає в розбитті обладнання на індивідуальні деталі та системи, кожній з яких надається конкретне значення залежно від її впливу на функціонування повітряного судна.

Кожна деталь чи система оцінюється за їхнім впливом на безпеку та ефективність повітряного транспорту. За результатами цієї оцінки розробляється графік планування технічного обслуговування, в якому враховуються інтервали обслуговування для кожної окремої деталі чи системи.

МВТ допомагає вчасно виявляти та усувати можливі проблеми з технічним обладнанням повітряних суден, сприяючи збереженню їхньої надійності та безпеки. Застосування цього методу дозволяє ефективно враховувати важливість кожної деталі та системи, забезпечуючи оптимальне технічне обслуговування з урахуванням стратегічних цілей повітряного транспорту. Такий підхід сприяє підвищенню довговічності обладнання та зниженню ймовірності аварій.

Евристика дозволяє знайти наближене рішення за короткий проміжок часу та аналізувати багатофакторні залежності між різними параметрами. Наприклад, за допомогою евристичного методу можна швидко визначити оптимальний час для проведення технічного обслуговування, враховуючи наявні ресурси та міжнародні розклади польотів. Крім того, евристичний метод може пропонувати розв'язання проблем, які не завжди зручно формалізувати. Наприклад, у випадку непередбачених змін у розкладі польотів або вмикання тимчасового резервного повітряного судна, евристичний метод може прискорити планування та забезпечити швидку адаптацію розкладу виконання робіт. Однак, необхідно враховувати деякі обмеження евристичного методу. Основна ідея цього методу полягає в тому, щоб вибирати та виконувати завдання в порядку їхнього найменшого часу виконання. Такий підхід спрямований на мінімізацію часу обробки загального потоку завдань.

Завдання складання розкладу виконання робіт є складним і має багато незрівнянних факторів, таких як обмеження польотних годин, доступність обладнання і робочої сили, технічні характеристики повітряних суден тощо. Тому, при застосуванні евристичного методу до планування технічного обслуговування повітряних суден, необхідно бути уважним до граничних значень та збалансовувати різні фактори при прийнятті рішення.

Можливість застосування евристичного методу до планування технічного обслуговування повітряних суден пропонує багато переваг в порівнянні з традиційними методами.

Швидкість обробки даних, здатність до адаптації та прийняття рішень, а також урахування непередбачених змін є головними перевагами евристичного методу. Однак, необхідно враховувати складність завдання та багатофакторні залежності при прийнятті рішень. Незважаючи на це, застосування евристичного методу до планування технічного обслуговування повітряних суден має великий потенціал.

Технічне обслуговування вимагає високоорганізованого та ефективного планування для забезпечення оптимальної експлуатації та безпеки польотів.

Одним із підходів до вирішення цього завдання є застосування евристичних методів у складанні розкладу виконання робіт, тобто застосування евристичного методу, зокрема метод виконання найбільш важливого завдання першим (*Weighted Shortest Job First*) для оптимізації процесу технічного обслуговування повітряних суден.

Евристичний метод базується на ідеї пріоритезації завдань в залежності від їх тривалості виконання. Завдання вимірюються а пріорітетом, та ззалежно від сортування розміщуються в розкладі перед завданнями з більшим часом виконання.

Планування технічного обслуговування обслуговування повітряних суден можна зобразити процесом бакового обслуговування. Повітряні судна вимагають регулярного і компетентного технічного обслуговування для забезпечення безпеки польотів та максимальної ефективності їхньої експлуатації.

Одним із ключових етапів процесу технічного обслуговування повітряних суден є бакове обслуговування, яке включає в себе догляд за паливними системами та баками повітряних суден. Заправлення літака входить у склад технічного обслуговування літаків і відноситься до бакового обслуговування. Це включає в себе заправлення паливом, а також контроль і обслуговування систем пального.

Бакове обслуговування важливий пункт в авіаії для забезпечення безпеки та ефективності польотів, оскільки правильно інтегрована система пального є критичною для операцій літака.

Процес заправлення літака включає у себе постачання палива з відповідних зберігальних танків аеропорту до баків літака. Технічний персонал відповідає за виконання цих операцій, перевіряючи правильність кількості пального, яке необхідно додати, і впевнюючись в правильності процедур забезпечення безпеки під час заправлення.

Бакове обслуговування літака відноситься до поточного технічного обслуговування (ТО). Поточне ТО включає в себе регулярні перевірки і обслуговування, щоб забезпечити безпеку і ефективність літака під час його експлуатації.

Зокрема, бакове обслуговування охоплює дії, пов'язані із зберіганням та постачанням пального, перевірку систем пального, а також заправлення літака перед кожним польотом. Бакове обслуговування літака включає в себе ряд технологічних операцій для правильного і безпечного заправлення літака пальним. Послідовність технологічних операцій при баковому обслуговуванні:

1. Припинення руху літака та його припаркування на визначеному майданчику для обслуговування;
2. Встановлення засобів безпеки, щоб забезпечити стабільність літака під час обслуговування;
3. Перевірка рівня пального в баках літака за допомогою відповідних індикаторів чи датчиків;
4. Заправлення літака необхідною кількістю пального за допомогою паливозаправних шлангів та систем нафтозаправочної апаратури;
5. Візуальна перевірка баків та системи паливопостачання на наявність витоків чи інших дефектів;
6. При необхідності, обслуговування паливного фільтра.

Всі ці технологічні операції займають певну, часто неточну, кількість часу, яку неможливо в деяких ситуаціях чітко визначити.

Саме в цьому випадку евристичний алгоритм може з мінімальною кількістю вхідних данних обробити і видати графік оптимального вирішення задачі.

Для дослідження евристичного алгоритму, а саме можливості використання алгоритму загалом в сфері технічнго обслуговування повітряних суден, а саме для створення графіку планування бакового обслуговування повітряних суден.

Для простого прикладу оберемо 3 странції техобслуговування аеропорту, або ж паливні бригади, та 10 літаків з своїми особистими годинами та властивостями для обслуговування. Використаємо вже обраний евристичний алгоритм для вирішення задачі.

Виконавці (верстати): 3 Бригади з паливозаправниками.

Задачі (рейси):

1) Рейс "Варшава – Париж" (прибуття в 10:25, виліт в 11:00);

2) Рейс "Варшава – Лондон" (прибуття в 11:15. виліт в 11:50);

3) Рейс "Варшава – Копенгаген" (прибуття в 11:00, виліт в 11:35);

4) Рейс "Варшава – Берлін" (прибуття в 11:45, виліт в 12:20);

5) Рейс "Варшава – Амстердам" (прибуття в 12:20, виліт в 12:55);

6) Рейс "Варшава – Рим" (прибуття в 12:30, виліт в 13:05);

7) Рейс "Варшава – Мадрид" (прибуття в 13:10, виліт в 13:55);

8) Рейс "Варшава – Відень" (прибуття в 13:45, виліт в 14:20);

9) Рейс "Варшава – Брюссель" (прибуття в 14:00, виліт в 14:40);

10) Рейс "Варшава – Загреб" (прибуття в 14:30, виліт в 15:10).

Спочатку вказуємо кількість паливо заправників (*workers*) та орієнтовний час на заправлення кожного літака . Вводимо інформацію про рейси, включаючи маршрут і час вильоту. Створюємо масиви для зберігання робочих паливо заправників, впорядкованих рейсів та часу роботи кожного паливо заправника. Алгоритм сортує рейси за часом вильоту для обробки їх в порядку. Для кожного рейсу алгоритм обирає заправника з найменшим часом роботи. Сортується список з часу початку заправлення як максимум із часу завершення роботи паливо заправника та часу вильоту рейсу. Оновлюється час завершення роботи паливо заправника та додається інформацію про рейс для обраного паливо заправника. Виводиться інформація про кожного паливо заправника, час виконання його задачі, виведення деталей про заправлення для кожного рейсу. Для кожного паливозаправника алгоритм виводить час завершення його роботи.

Результат виконання евристичного алгоритму планування зображено на рис. 2.2.

```
Worker 1: Refueling Warsaw-Paris at 10:25 - 10:45
Worker 1: Refueling Warsaw-Berlin at 11:45 - 12:05
Worker 1: Refueling Warsaw-Madrid at 13:10 - 13:30
Worker 1: Refueling Warsaw-Zagreb at 14:30 - 14:50
Worker 2: Refueling Warsaw-Copenhagen at 11:00 - 11:20
Worker 2: Refueling Warsaw-Amsterdam at 12:20 - 12:40
Worker 2: Refueling Warsaw-Vienna at 13:45 - 14:05
Worker 3: Refueling Warsaw-London at 11:15 - 11:35
Worker 3: Refueling Warsaw-Rome at 12:30 - 12:50
Worker 3: Refueling Warsaw-Brussels at 14:00 - 14:20
```

Рис. 2.2. Результат виконання алгоритму

Підхід евристичного алгоритму дозволяє прискорити процес планування, оскільки він проглядає всю базу даних та обирає задачу прямо для конкретного запису. Структурна схема планування задач алгоритму представлена на рис. 2.3.

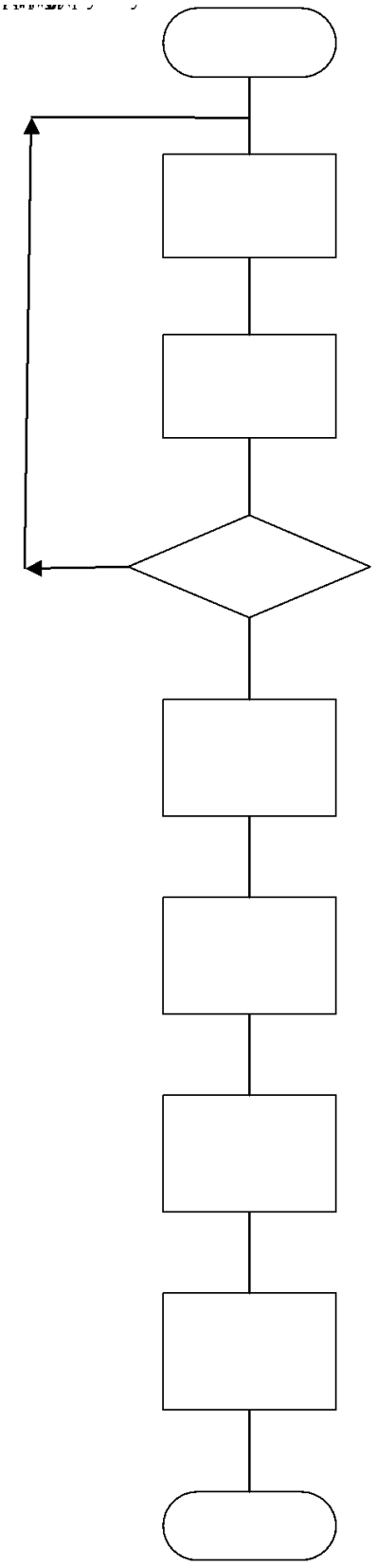

Рис. 2.3. Схема алгоритму планування задач

Евристичний алгоритм планування розкладу дозволяє оптимально розподілити завдання (заправлення рейсів) між паливо заправниками, враховуючи час вильоту кожного рейсу та забезпечуючи ефективне використання часу.

Модуль може використовуватися для автоматизації процесів планування, виконання та моніторингу технічного обслуговування повітряних суден, що сприяє підвищенню безпеки та зниженню витрат на експлуатацію.

Проаналізовано та доведено можливість застосування планування технічного обслуговування повітряних суден евристичного методу складання розкладу виконання робіт. В контексті планування технічного обслуговування повітряних суден, евристичний метод може бути застосований для складання розкладу виконання робіт за допомогою інформації про доступні ресурси, розклади польотів та інші параметри. Однією з головних переваг евристичного методу в порівнянні з традиційними методами є його швидкість та можливість обробки великих обсягів даних. Цей метод ефективний в ситуаціях, коли тривалість виконання завдань визначається однозначно і не змінюється в процесі виконання.

2.3. Порівняння ефективності евристичного методу з іншими стратегіями планування технічного обслуговування повітряних суден

Технічне обслуговування повітряних суден вимагає виваженого вибору стратегії для забезпечення їх безпеки, надійності та продуктивності. Один із підходів до цього завдання - використання евристичного методу. Щоб з'ясувати його ефективність, порівняємо його з іншими популярними стратегіями планування технічного обслуговування. Кожна стратегія має свої переваги та недоліки. Евристичний метод вирізняється високою ефективністю та гнучкістю, але вимагає експертного досвіду. Вибір оптимальної стратегії повинен бути здійснений з урахуванням конкретних умов та вимог експлуатації повітряних суден.

Щоб порівняти ефективність між стандартними методами планування обслуговування та евристичними методами було створено таблицю 2.1.

Таблиця 2.1

Порівняння ефективності методів

Критерії /Методи	Евристичний метод	Регулярний план тех обслуговування	Прогнозування на основі стану	Запланована заміна обладнання
Ефективність	Висока	Середня	Висока	Висока
Простота та Передбачуваність	Висока	Висока	Середня	Середня
Мінімізація Витрат	Висока	Середня	Низька	Середня
Відповідність Стандартам	Середня	Висока	Висока	Висока
Аварійна Готовність	Висока	Середня	Висока	Середня

Важливо врахувати, що однією з ключових переваг евристичного методу є його висока швидкість прийняття рішень. У порівнянні з математичними стратегіями, такими як оптимізація лінійних програм, евристика в здатна давати прийнятні рішення за значно менший час. Це особливо важливо в сфері технічного обслуговування повітряних суден, де час часто є критичним фактором.

Порівнюючи жадібний евристичний алгоритм із стандартним технічним обслуговуванням, можна враховувати кілька ключових відмінностей та переваг кожного підходу.

Стандартне лінійне технічне обслуговування може використовувати більш розгорнуті стратегії та планування, такі як методи визначення пріоритетів, оптимізації загального часу виконання, розподіл завдань з урахуванням різних критеріїв та врахуванням обмежень ресурсів.

Однак важливою перевагою жадібного методу є його простота та легкість реалізації. Він може бути ефективним у випадках, коли головною метою є мінімізація часу виконання. З іншого боку, стандартне лінійне технічне

обслуговування може бути більш гнучким та адаптивним до різних умов та вимог, оскільки дозволяє враховувати більше факторів при плануванні. Стандартні ж методи технічного обслуговування можуть бути більш складними у впровадженні та управлінні, а також вимагати більше ресурсів для аналізу та планування. Жадібний метод, з іншого боку, може бути менш гнучким у вирішенні комплексних сценаріїв та вимагати певного рівня узагальнення.

Евристичний метод може бути ефективним у простих сценаріях, де головною метою є мінімізація часу виконання. З іншого боку, стандартні лінійні методи технічного обслуговування можуть надати більше гнучкості та адаптивності при вирішенні більш складних завдань та врахуванні різних критеріїв оптимізації. При порівнянні обох методів слід також враховувати витрати та ресурси, необхідні для впровадження та підтримки кожного з них. Жадібний метод, завдяки своїй простоті, може виявитися ефективнішим з точки зору вартості реалізації та управління, особливо у невеликих масштабах. У той час як стандартні лінійні методи можуть вимагати великого обсягу ресурсів для складного планування та впровадження, вони зазвичай можуть надати більше можливостей для адаптації до змін у вхідних умовах або вимогах.

Оцінка ситуації, використання експертних знань та здатність враховувати нелінійність у завданнях планування технічного обслуговування роблять евристичний метод ефективним при різноманітних умовах експлуатації. Також можна виділити відсутність потреби в значних обчислювальних ресурсах у порівнянні з деякими математичними стратегіями, які вимагають значних обчислювальних ресурсів, евристичний метод може працювати ефективно на менших обчислювальних потужностях. Ці аспекти роблять його доступним і більш економічно вигідним для аеропортів в певних умовах. Такий підхід може допомагати забезпечувати ефективне технічне обслуговування повітряного флоту за обмеженими ресурсами. Система може швидко адаптуватися до змінних умов експлуатації та враховувати різноманітні фактори, такі як інтенсивність експлуатації.

Евристичний є корисним з точки зору реалізації, адже через відсутність складних математичних моделей спрощує процес впровадження та підтримки, що є важливим фактором у практичних застосуваннях. Евристичний метод дає можливість враховувати експертні знання та досвід фахівців, що робить його особливо цінним в областях, де людський досвід важливий. У контексті технічного обслуговування повітряних суден, це дозволяє враховувати унікальні аспекти конкретної технічної системи та умов її експлуатації.

Евристичний метод виявляється дуже ефективним у плануванні технічного обслуговування повітряних суден завдяки своїм унікальним перевагам, які включають швидкість прийняття рішень, гнучкість до змінних умов, економію обчислювальних ресурсів, простоту реалізації та можливість врахування експертних знань. Метод може бути ефективним у випадках, коли обсяг доступних даних обмежений. Одна з особливостей евристичного підходу полягає в тому, що він може працювати навіть коли вхідні дані не є повними чи точними. Це робить його особливо корисним у ситуаціях, де зібрання повних та точних даних може бути викликом. Евристичний метод дозволяє легко внести зміни в стратегію без необхідності перегляду складних математичних рівнянь чи алгоритмів. Це забезпечує зручність в апгрейді та адаптації до нових вимог або змінних умов експлуатації повітряних суден.

У сфері авіаційного технічного обслуговування часто виникають невизначеності, такі як випадкові збої чи непередбачені умови експлуатації. Евристичний метод проявляє себе добре в умовах невизначеності, забезпечуючи стійкість та ефективність при вирішенні завдань планування обслуговування.

Загалом, ці позитивні аспекти разом із зазначеними раніше особливостями роблять евристичний метод важливим і привабливим вибором для практичного впровадження у сфері планування технічного обслуговування повітряних суден.

2.4. Висновки до розділу

У даном розділі були розглянуті ключові компоненти технології планування технічного обслуговування повітряних суден, зокрема, комп'ютеризована система управління технічним обслуговуванням та програмне забезпечення для прогнозного технічного обслуговування. Акцент був зроблений на важливій ролі цих технологій у забезпеченні безпеки та ефективності експлуатації повітряних суден. Ці системи дозволяють авіакомпаніям і організаціям з технічного обслуговування ефективно керувати ресурсами, планувати та виконувати технічне обслуговування, мінімізуючи незаплановані простої та ризики аварій.

Зазначено, що ці системи відстежують та оптимізують всі аспекти технічного обслуговування, включаючи планування, робочі замовлення, управління запасами та історію обслуговування. Це призводить до мінімізації незапланованих простоїв та ризиків аварій, сприяючи загальній ефективності авіаційних операцій.

Порівняльний аналіз різних методів планування технічного обслуговування, таких як метод визначення залишкового ресурсу та метод відновлювального технічного обладнання, вказує на їхню важливість у використанні ресурсів та уникненні непланованих відмов.

Евристичний метод, а саме метод виконання найбільш важливого завдання першим, був представлений як потенційно ефективний для планування технічного обслуговування повітряних суден. Його основна перевага полягає в оптимізації процесу технічного обслуговування, враховуючи важливість та тривалість завдань. Зазначено, що цей метод може бути особливо корисним у швидкому реагуванні на зміни в розкладі польотів та забезпеченні стратегічного планування.

Було проаналізовано та розглянуто різні методи планування технічного обслуговування, включаючи метод визначення залишкового ресурсу та метод відновлювального технічного обладнання. Важливість цих методів полягає в їх здатності ефективно використовувати ресурси та уникати непланових збоїв.

Досліджено евристичний метод, а саме метод виконання найбільш важливого завдання першим, який може бути застосований для планування технічного обслуговування повітряних суден. Цей метод використовується для оптимізації процесу технічного обслуговування, надаючи перевагу завданням в залежності від їхнього важливого значення та тривалості виконання.

В перспективі подальших досліджень, зосереджених на методі виконання найбільш важливого завдання першим у контексті технічного обслуговування повітряних суден, можна сподіватися на вдосконалення його ефективності та адаптації до реальних умов авіаційного обслуговування. Аналіз цього методу у відповіді на зміни в розкладі польотів та забезпечення стратегічного планування наголошує на його потенційних перевагах у швидкій обробці даних та динамічній адаптабельності.

Підсумовуючи можна зазначити що евристичний метод має потенціал для застосування в плануванні технічного обслуговування, забезпечуючи швидкість обробки даних та адаптивність до змін у розкладі польотів. Однак, необхідно ураховувати складність завдання та багатофакторні залежності при використанні евристичного методу.

РОЗДІЛ 3
ПРОГРАМНА РЕАЛІЗАЦІЯ МОДУЛЯ ПЛАНУВАННЯ ТЕХНІЧНОГО ОБСЛУГОВУВАННЯ ПОВІТРЯНИХ СУДЕН

3.1. Інформаційне забезпечення модуля

В першу чергу необхідно описати технології, які були використані для розробки системи.

Програма буде писатися мовою програмування *C#* у середовищі програмування *Visual Studio 2022*. *C#* – проста, потужна, статично типізована, об'єктно орієнтована мова програмування від компанії *Microsoft*. *C#* входить до сімейства мов програмування *C*.

C# – строго типізована об'єктно-орієнтована мова програмування. *C#* є мовою програмування з відкритим кодом, простим, сучасним, гнучким і універсальним. *C#* – це мова програмування, розроблена та запущена корпорацією Майкрософт у 2001 році. *C#* – проста, сучасна та об'єктно-орієнтована мова програмування. Мета *C#* полягала в тому, щоб розробити мову програмування, яка не тільки проста у вивченні, але й підтримує сучасні функціональні можливості для розробки будь-якого програмного забезпечення.

Однак мова *C#* була розроблена, щоб враховувати потреби бізнесу та підприємств. Мова *C#* була розроблена для компаній, щоб створювати всі види програмного забезпечення за допомогою однієї мови програмування.

C# забезпечує функціональність для підтримки сучасної розробки програмного забезпечення. *C#* підтримує потреби розробки веб-сайтів, мобільних пристроїв і програм. Деякі з сучасних функцій мови програмування, які підтримує *C#*, це генерики, типи змінних, автоматична ініціалізація типів і колекцій, лямбда-вирази, динамічне програмування, асинхронне програмування, кортежі, зіставлення шаблонів, розширене налагодження та обробка винятків тощо

На синтаксис мови *C#* вплинули *C++, Java, Pascal* та деякі інші мови, які легко прийняти. *C#* також уникає складності та неструктурованих функцій мови.

Перша версія мови *C#* була створена в 1998-2001 роках, групою інженерів *Microsoft* під керівництвом Андреса Гейлсберга та Скотта Вільтаумота, як основна мова програмування платформи *Microsoft .Net. C#* увібрав в себе найкращі властивості попередників – мов *C, C++, Modula, Object Pascal*, спираючись на практичний досвід їх використання. Деякі проблематичні моделі, що до цього використовувались у мовах програмування, зокрема множинне спадкування класів(яке використовується у мові *C++)*, були свідомо виключені.

В багатьох відношеннях мова *C#* дуже схожа на *Java*, це відображено в синтаксисах та основних поняттях цих мов програмування.

Назва мови *C#* трактується як наступне покоління розвитку *C++*, а символ # – символізує *"++++"*. Спочатку в назві фігурував "♯" (англійською *sharp*), однак через відсутність цього символу на клавіатурі, використовується знак для позначення номеру *"#"*. Загалом назву мови можна позначати обома символами.

Розробка мови *C#* розпочалася в грудні 1998 року, та готувався до випуску разом з продуктами групи *Millenium.* Прокт мав назву *COOL(C-style Object Oriented Language)*, та розроблявся як аналог *Java* від компанії *Oracle*. *C#* був анонсований, широкому загалу, в 2000 році, як основна мова платформи *Microsoft .Net Framework*. В цьому ж році з'явилася перша загальнодоступна бета-версія.

Перша фінальна версія мови програмування *C#* була випущена в 2002 році разом з середовищем інтегрованої розробки програмного забезпечення *Visual Studio .Net. C#* вважається простою мовою. Її компактний код легко читати, що надзвичайно зручно для оптимізації командної розробки програмного забезпечення. *C#* працює на платформі *.NET*, яка вважається надійною та добре спроєктованою.

C# може заощадити час, оскільки ця мова була розроблена, щоб полегшити створення на її основі потужних інструментів.

Мова програмування *C#* масштабована і проста в обслуговуванні. Це мова з відкритим вихідним кодом.

C# є відкритим вихідним кодом у рамках *.NET Foundation*, яка керується та працює незалежно від *Microsoft*. Специфікації мови *C#,* компілятори та пов'язані інструменти є проектами з відкритим кодом на *Github.* Хоча розробкою функцій мови *C#* керує *Microsoft*, співтовариство з відкритим кодом дуже активно займається розробкою та вдосконаленням мови.

C# є швидкою порівняно з кількома іншими мовами програмування високого рівня. *C#* – кросплатформна мова програмування. Вона дозволяє створювати програми *.NET*, які можна розгортати на платформах *Windows, Linux і Mac*. Програми *C#* також можна розгортати в хмарі та контейнерах.

C# є типово безпечною мовою. *C#* не дозволяє перетворення типів, які можуть призвести до втрати даних або інших проблем. *C#* дозволяє розробникам писати безпечний код. *C#* також зосереджується на написанні ефективного коду.

У *C#* підтримуються типи, що допускають значення *Null* і не допускають значення *Null*.

Оголошення структури тільки для читання, щоб вказати, що тип є незмінним і дозволяє компілятору зберігати копії під час використання параметрів in.

Якщо розмір структури лише для читання більший за *IntPtr.Size*, важливо його передати його як параметр *in* з міркувань продуктивності.

Структура як параметр *in*, якщо вона не оголошена з модифікатором *readonly* може негативно вплинути на продуктивність і призвести до незрозумілої поведінки.

Важливим є використання структури посилання або структури посилання лише для читання, наприклад *Span<T> або ReadOnlySpan<T>*, щоб працювати з пам'яттю як послідовністю байтів.

Стандарт *C#* підтримується як міжнародним стандартом (*ISO/IEC 23270:2018) і ECMA (ECMA-334).*

C# об'єднує елементи синтаксису *C і C++,* роблячи його відносно легким для програмістів, знайомих з цими мовами.

Введення асинхронного програмування з ключовими словами *async* і *await* робить обробку багатозадачності більш зручною.

C# можна використовувати для розробки будь-яких програм, включаючи клієнтські програми *Windows*, компоненти та бібліотеки, служби та *API*, веб-програми, мобільні програми, хмарні програми та відеоігри.

C# може бути використаний для створення:

– Клієнтські програми *Windows*;

– Бібліотеки та компоненти *Windows*;

– Служби *Windows*;

– Веб-додатки;

– Веб-сервіси та *веб-API*;

– Власні мобільні програми для *iOS* і *Android*;

– Бекенд-сервіси;

– Хмарні програми та служби *Azure*;

– База даних із використанням інструментів *ML/Data*;

– Програмне забезпечення для взаємодії, таке як *Office, SharePoint*;

– Штучний інтелект і машинне навчання;

– Блокчейни та технологія розподіленої книги, включаючи криптовалюту;

– Пристрої Інтернету речей (*IoT*);

– Ігрові приставки та ігрові системи;

– Відео ігри.

Існує велика спільнота розробників *C#*, до якої можна долучитися, щоб поставити питання, надати відповідь чи організувати мозковий штурм. *C#* має блискучі перспективи завдяки популярності, універсальності та наявним програмним продуктам, створеним з використанням цієї мови.

.NET підтримує два типи колекцій: загальні колекції та незагальні колекції. До *.NET 2.0* це були лише колекції, а коли до *.NET* додалися генерики, додалися й колекції генериків.

З використанням *C Sharp* написані наступні популярні програми та додатки: *Paint.NET, Microsoft Visual Studio, Windows Installer XML, Pinta* та інші. Також деякі ігри, зокрема: *Pokemon Go, Cuphead, Hearthstone, Subnautica, Rimworld* тощо.

Окрім *Microsoft*, *C#* використовують ряд інших відомих у світі компаній та корпорацій. Мова йде про *Google, Softserve, Amazon, Oracle, Siemens, Tesco* та інші.

Основні переваги мови програмування *C#*:

– використання надійної платформи *.NET,* яка постійно розвивається;

– строга статична типізація, що запобігає виникненню помилок та робить код більш безпечним;

– наявність великої спільноти розробників, де можна швидко отримати відповіді на професійні питання;

– використання *Visual Studio* з широким функціоналом для створення застосунків, консольних програм, програм з графічним інтерфейсом;

– доступна структурована документація;

– універсальність застосування: розробка як стандартних програм, так і сучасних застосунків.

Варто зазначити, що *C Sharp* включає в себе переваги *C++* і *Java*, тим самим здобувши популярність серед розробників. Вона також найбільше відповідає стандарту *Common Language Infrastructure* (*CLI* - інтерфейс для взаємодії користувача з операційною системою).

Загальна мовна інфраструктура (*CLI*) забезпечує мовно-нейтральну платформу для розробки та виконання програм. Завдяки реалізації основних аспектів *.NET Framework* у межах *CLI* ці функції не будуть прив'язані до однієї мови, а будуть доступні багатьма мовами, підтримуваними платформою.

.NET Framework містить загальномовне середовище виконання (*CLR*). Він служить механізмом виконання *.NET Framework* і пропонує багато послуг, таких як керування пам'яттю, безпека типів, обробка винятків, збирання сміття, безпека та керування потоками. Усі програми, написані для *.NET Framework*, виконуються *CLR*.

Програми, написані для *.NET Framework*, компілюються в код *Common Intermediate Language (CIL)*, а не безпосередньо компілюються в машинний код. Під час виконання специфічний для архітектури своєчасний компілятор (*JIT*) перетворює код *CIL* на машинний код.

Формально *.NET* – це платформа розробника з відкритим кодом, створена *Microsoft* для створення багатьох різних типів програм. Доступно писати програми *.NET на C#, F#, Visual C++* або *Visual Basic*.

Неофіційно *.NET* – це інструмент, який дозволяє створювати та запускати програми на *C#* Існує кілька версій *.NET*. Вони виконують однакову роботу, але призначені для різних операційних систем.

.NET Framework – це запатентована програмна основа, розроблена *Microsoft*, яка в основному працює в *Microsoft Windows*. Це була переважна реалізація спільної мовної інфраструктури (*CLI*), поки її не замінив кросплатформенний проект *.NET*. Він містить велику бібліотеку класів під назвою *Framework Class Library (FCL)* і забезпечує взаємодію мов (кожна мова може використовувати код, написаний іншими мовами) у кількох мовах програмування. Програми, написані для *.NET Framework*, виконуються в програмному середовищі (на відміну від апаратного середовища), яке називається *Common Language Runtime (CLR). CLR* — це віртуальна машина програми, яка надає такі служби, як безпека, керування пам'яттю та обробка винятків. Таким чином, комп'ютерний код, написаний за допомогою *.NET Framework*, називається «керованим кодом». *FCL і CLR* разом складають *.NET Framework*.

FCL забезпечує інтерфейс користувача, доступ до даних, підключення до бази даних, криптографію, розробку веб-додатків, числові алгоритми та мережеві комунікації. Програмісти створюють програмне забезпечення, поєднуючи вихідний код із *.NET Framework* та іншими бібліотеками. Фреймворк призначений для використання більшістю нових програм, створених для платформи *Windows*.

.NET Framework починався як пропрієтарне програмне забезпечення, хоча фірма працювала над стандартизацією стека програмного забезпечення майже відразу, навіть до його першого випуску. Незважаючи на зусилля зі стандартизації, розробники, головним чином із спільнот вільного та відкритого програмного забезпечення, висловили своє занепокоєння вибраними умовами та перспективами будь-якої вільної реалізації з відкритим кодом, особливо щодо патентів на програмне забезпечення.

Відтоді корпорація Майкрософт змінила розробку *.NET*, щоб більш точно слідувати сучасній моделі проекту програмного забезпечення, розробленого спільнотою, включно з випуском оновлення свого патенту, яке обіцяє вирішити проблеми.

.NET Core, як відкрита та крос-платформена платформа розробки від *Microsoft*, представляє сучасний та гнучкий фреймворк, що відзначається рядом ключових характеристик, роблячи його привабливим для розробників у різних сферах та сценаріях. *.NET* Core вирізняється своєю крос-платформеністю, що дозволяє розробникам створювати додатки, які працюватимуть на різних операційних системах без значних модифікацій у вихідному коді. Це робить фреймворк універсальним і готовим до використання в різних середовищах.

Надзвичайно важливою рисою *.NET Core* є його статус як відкритого вихідного коду. Відкритість сприяє взаємодії та співпраці розробників, які можуть не лише використовувати фреймворк, а й вносити свої власні покращення та модифікації.

Важливо відзначити, що *.NET Core* підтримує кілька мов програмування, включаючи потужну мову *C#*, що робить його дуже доступним для широкого кола розробників. Спільно з цим, вбудована бібліотека фреймворку надає зручні інструменти для вирішення різноманітних задач без необхідності розроблення власних рутин.

Ще однією значущою характеристикою є вбудований фреймворк *ASP.NET Core*, спрямований на розробку веб-додатків та служб. Висока продуктивність та можливість роботи на різних платформах роблять його відмінним вибором для сучасних веб-розробок.

.NET Core підтримує мікросервісну архітектуру, що робить його ідеальним для розробки розподілених та масштабованих додатків.

Підхід розподілених та масштабованих додатків особливо актуальний у контексті сучасних вимог до розподіленого програмного забезпечення.

Окрім того, *.NET Core* інтегрується з іншими інструментами від *Microsoft*, зокрема, з середовищем розробки *Visual Studio*, надаючи розробникам зручне та продуктивне середовище для творчості.

Не менш важливою є можливість використання *.NET Core* для розробки мобільних додатків за допомогою *Xamarin*. Це розширює область застосування фреймворку на платформи *Android* та *iOS*.

Загальний висновок полягає в тому, що *.NET Core* є потужним та гнучким інструментом для розробки різноманітних додатків, що враховує вимоги сучасного програмного забезпечення. Його відкритий вихідний код, крос-платформеність та великий функціонал роблять його важливим гравцем у світі розробки програмного забезпечення. У майбутньому, дослідження та вдосконалення фреймворку можуть призвести до ще більш широкого застосування та покращення його можливостей.

.NET Framework містить реалізацію базових стандартних бібліотек *CLI*. Бібліотека класів *.NET Framework (FCL)* організована в ієрархії просторів імен. Більшість вбудованих інтерфейсів прикладного програмування (*API*) є частиною просторів імен *System.** або *Microsoft.**. Ці бібліотеки класів реалізують багато загальних функцій, таких як читання та запис файлів, рендеринг графіки, взаємодія з базою даних і маніпулювання документами *XML*. Бібліотеки класів доступні для всіх *CLI*-сумісних мов. *FCL* реалізує бібліотеку базових класів *CLI (BCL)* та інші бібліотеки класів — деякі визначені *CLI*, а інші — специфічні для *Microsoft.*

BCL включає невелику підмножину всієї бібліотеки класів і є основним набором класів, які служать основним *API CLR*. Для *.NET Framework* більшість класів, які вважаються частиною *BCL*, знаходяться в *mscorlib.dll*, *System.dll* і *System.Core.dll*. Класи *BCL* доступні в *.NET Framework*, а також в альтернативних реалізаціях *CLI*, включаючи *.NET Compact Framework*, *Microsoft Silverlight, .NET Core і Mono.*

FCL відноситься до всієї бібліотеки класів, яка поставляється з *.NET* Framework. Він включає *BCL*, розширений набір бібліотек, включаючи *Windows Forms, ASP.NET* і *Windows Presentation Foundation (WPF)*.

Також він має розширення для бібліотек базових класів *ADO.NET*, *Language Integrated Query (LINQ)*, *Windows Communication Foundation (WCF)*.) і *Workflow Foundation (WF)*. *FCL* набагато більший за обсягом, ніж стандартні бібліотеки для таких мов, як *C++*, і порівнянний за обсягом зі стандартними бібліотеками *Java.*

Хоча *Microsoft* ніколи не впроваджувала повну структуру в будь-якій системі, окрім *Microsoft Windows*, вона розробила таку структуру, щоб бути кросплатформенною, і реалізації доступні для інших операційних систем. Microsoft подала специфікації для *CLI* (що включає бібліотеки базових класів, *CTS і CIL*), як *Ecma International (ECMA)*, так і *International* Організація зі стандартизації (*ISO*), що робить їх доступними як офіційні стандарти. Це дає змогу третім сторонам створювати сумісні реалізації фреймворку та його мов на інших платформах. *Core* кросплатформенний *.NET (раніше .NET Core)* також офіційно доступний для багатьох дистрибутивів *Linux і MacOS*.

.NET Framework включає збирач сміття (*GC*), який періодично запускається в окремому від потоку програми потоку, який перераховує всі непридатні об’єкти та відновлює виділену для них пам’ять. Це недетермінований збирач сміття, що стискає, позначає та підмітає. *GC(garbage collector)* запускається лише тоді, коли використовується встановлений обсяг пам’яті або в системі достатньо пам’яті. Оскільки не гарантується, коли умови для повернення пам’яті досягнуті, запуски бирача сміття є недетермінованими. Кожна програма *.NET* має набір коренів, які є покажчиками на об’єкти в керованій купі (керовані об’єкти). Вони включають в себе посилання на статичні об'єкти, об'єкти, визначені як локальні змінні або параметри методу, які зараз знаходяться в області видимості, і об'єкти, на які посилаються регістри ЦП.

Коли процес звільнення памяті запускається, він призупиняє роботу програми, а потім для кожного об’єкта, на який посилається в корені, рекурсивно перераховує всі об’єкти, доступні з кореневих об’єктів, і позначає їх як доступні. Він використовує метадані *CLI* та відображення, щоб виявити об’єкти, інкапсульовані об’єктом, а потім рекурсивно пройти їх. Потім він перераховує всі об’єкти в купі (які

спочатку були розміщені безперервно) за допомогою відображення. Усі об’єкти, не позначені як доступні, є сміттям.

Оскільки пам’ять, яку містить сміття, не має значення, вона вважається вільним простором. Однак це залишає шматки вільного простору між об’єктами, які спочатку були суміжними. Потім об’єкти стискаються разом, щоб вільний простір у керованій купі знову став суміжним. Будь-яке посилання на об’єкт, визнане недійсним через переміщення об’єкта, оновлюється збір сміття, щоб відобразити нове розташування.Додаток відновлюється після завершення збору сміття. Остання версія *.NET Framework* використовує паралельне збирання сміття разом із кодом користувача, що робить паузи непомітними, оскільки це робиться у фоновому режимі.

.NET Framework була переважною реалізацією *CLI* до випуску *.NET.* Існують інші реалізації частин фреймворку. Хоча механізм виконання описується специфікацією *ECMA-ISO,* інші його реалізації можуть бути обтяжені патентними проблемами. Крім того, частини *FCL* мають специфічні для *Windows* функції та поведінку, тому впровадження на платформах, відмінних від *Windows*, може бути проблематичним.

Деякими альтернативними реалізаціями частин фреймворку можуть бути:

1. *.NET Micro Framework* — це платформа *.NET* для пристроїв із надзвичайно обмеженими ресурсами. Він містить невелику версію *CLR* і підтримує розробку на *C# (*хоча деякі розробники мали змогу використовувати *VB.NET* , хоча й із певною кількістю злому та з обмеженими можливостями) і налагодження (в емуляторі чи на апаратному забезпеченні), обидва за допомогою *Microsoft Visual Studio*. Він також містить підмножину бібліотеки класів *.NET Framework* (близько 70 класів із приблизно 420 методами), структуру графічного інтерфейсу користувача, яка частково базується на *WPF*, і додаткові бібліотеки, специфічні для вбудованих програм;
2. *Mono* є реалізацією *CLI* та *FCL* і надає додаткові функції. Воно ліцензовано як безкоштовне програмне забезпечення за ліцензією *MIT*. Він включає

підтримку бібліотек *ASP.NET*, *ADO.NET і Windows Forms* для широкого діапазону архітектур і операційних систем. Він також включає компілятори *C#* і *VB.NET*;

3. *Portable.NET* (частина *DotGNU*) забезпечує реалізацію *CLI*, частини *FCL* і компілятор *C#.* Він підтримує різноманітні процесори та операційні системи. Проект було припинено, останній стабільний випуск був у 2009 році;
4. *Microsoft Shared Source Common Language Infrastructure* — це реалізація *CLR*. Однак остання версія працює лише на *Windows XP SP2* і не оновлювалася з 2006 року. Таким чином, вона не містить усіх функцій версії 2.0 *.NET Framework*;
5. *CrossNet* є реалізацією *CLI* та частин *FCL*. Це безкоштовне програмне забезпечення з відкритим кодом.

M*icrosoft Visual Studio (VS)* - це інтегроване середовище розробки (*IDE*), розроблене компанією *Microsoft* для підтримки різних мов програмування та платформ розробки. *Visual Studio* надає розробникам зручні інструменти для написання коду, відлагодження програм, роботи з версіями, створення інтерфейсів користувача та інші завдання, пов'язані з розробкою програмного забезпечення.

Visual Studio підтримує широкий спектр мов програмування, включаючи *C#, C++, Visual Basic, F#, Python, JavaScript,* і багато інших. *Visual Studio* надає потужні засоби налагодження, такі як точки зупинки, вивчення значень змінних, відслідковування викликів функцій, аналіз стеку викликів та інші.

Інтегрована система керування версіями (*VCS*) дозволяє розробникам працювати з репозиторіями, такими як *Git* та *Team Foundation Server* (*TFS*), щоб відстежувати зміни в коді та спільно працювати з іншими розробниками. Є інструменти для візуального створення графічних інтерфейсів користувача для десктопних, веб та мобільних додатків. *Visual Studio* надає набір шаблонів проектів для різних типів додатків, що полегшує створення нових проектів з певним функціоналом.

Взаємодія з хмарами, зокрема з платформою *Azure*, для розгортання та управління хмарними службами. Засоби для написання та виконання тестів, включаючи модульні тести, тести одиниць, інтеграційні тести тощо. Великий вибір розширень, додатків та налаштувань, що дозволяють розробникам налаштувати робоче середовище під свої потреби.

Можливість використовувати *Visual Studio Code,* легший та більш зручний редактор коду, або *Visual Studio* для розробки. *Visual Studio* є потужним інструментом для розробників, який підтримує їх в усіх етапах процесу розробки програмного забезпечення.

Евристичні методи дозволяють здійснити розв’язання складних задач, для яких людина не може дати точний (математичний) алгоритм. Евристичні методи відтворюють логіку міркувань людини в процесі розв’язання конкретної задачі.

Однак, на відміну від традиційних методів, які здійснюють виведення рішень на основі моделей представлення знань про предметну область, евристичні методи не вимагають наявності таких моделей, а логіка міркувань людини відбивається (моделюється) самим алгоритмом.

Ця програма реалізує алгоритм планування завдань між працівниками, спрямований на рівномірне розподіл завдань для зменшення загального часу виконання. Спочатку користувач вводить кількість працівників та інформацію про завдання (тип та тривалість) у відповідних форматах. Кількість працівників перевіряється на коректність та позитивність. Якщо введення некоректне, програма виводить повідомлення про помилку та завершує роботу. Завдання також перевіряються на валідність та конвертуються у внутрішній формат (*task* об'єкти). Створюється черга завдань для кожного працівника та інші допоміжні структури. Завдання сортуються за спаданням тривалості, щоб спочатку обробляти довші завдання. Для кожного завдання обчислюється сумарний час виконання кожним працівником. Завдання призначається працівнику, у якого сумарний час виконання мінімальний. Після виконання алгоритму користувачеві виводиться інформація про розподіл завдань між працівниками, а також час виконання для кожного працівника, виводиться інформація про те, який працівник виконує кожне завдання, додатково

виводиться максимальний можливий сумарний час виконання серед усіх працівників.

3.2. Програмне забезпечення модуля

Так як мовою програмування була обрана *C#* з використанням *.Net,* найкращим вибором середовища програмування було обрано *Microsoft Visual Studio*. З проаналізованої та обдуманої інформації вище, було обрано створення застосунку для розрахунку оптимального планування робіт для технічного обслуговування повітряних суден між робочими завдякми використанню евристичного алгоритму.

Початок модулю йде з завантаження базових бібліотек. Далі йде фрагмент коду відповідає за введення користувачем кількості робітників, які будуть виконувати завдання.

Після введення дані перевіряються на коректність. Якщо введені дані некоректні, програма виведе повідомлення про помилку та завершить свою роботу.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Windows;
namespace TO
{
public partial class MainWindow : Window
{ private const double START_TIME = 720.0; // Початковий час 12:00
 public MainWindow()
{ InitializeComponent();
}
```

Цей фрагмент коду визначає клас *MainWindow* у просторі імен TO, який є частиною (*partial*) вікна *WPF (Window)*.

namespace TO: Визначає новий простір імен для класу *MainWindow* та інших пов'язаних класів чи об'єктів.

public partial class MainWindow : Window: Визначає клас *MainWindow*, який є частиною вікна *WPF (Window)*. Ключове слово *partial* вказує на те, що цей клас може мати свої частини у інших файлів.

private const double START_TIME = 720.0;: Оголошує константу *START_TIME*, яка представляє початковий час у хвилинах (12:00). Це використовується в алгоритмі планування завдань для забезпечення початку праці не раніше 12:00.

public MainWindow(): Конструктор класу *MainWindow*. Цей конструктор викликає конструктор базового класу *Window* (з якого вікно успадковане) і викликає метод *InitializeComponent()*. Метод *InitializeComponent()* генерується автоматично та містить код для ініціалізації компонентів інтерфейсу користувача з файлу *XAML*.

private void ScheduleTasks_Click(object sender, RoutedEventArgs)

{

if (!int.TryParse(txtNumberOfWorkers.Text, out int numberOfWorkers) || numberOfWorkers <= 0)

{

*lstResults.Items.Add("*Невірний ввід для кількості робітників. Будь ласка, введіть додатне ціле число.*");*

return;

}

}

private void ScheduleTasks_Click(object sender, RoutedEventArgs):

Вікно виконання програми з введенням даних зображено на рис. 3.1.

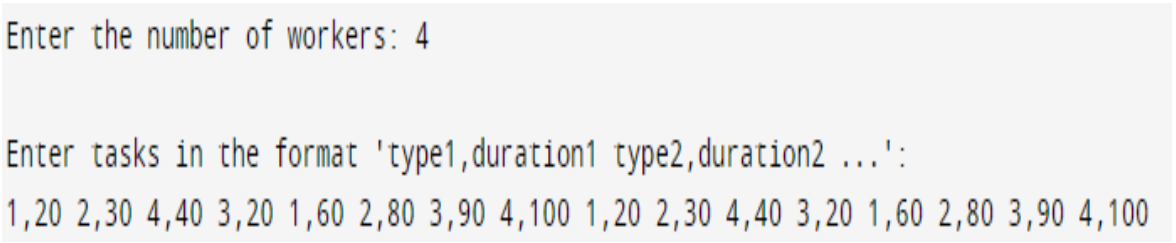

Рис. 3.1. Вікно виконання програми з введенням даних

Наступний метод, який буде викликаний при кліку на відповідну кнопку чи елемент у користувацькому інтерфейсі *WPF*. Параметри *object sender* та *RoutedEventArgs* вказують на те, що метод є обробником події та приймає інформацію про подію.

if (!int.TryParse(txtNumberOfWorkers.Text, out int numberOfWorkers) || numberOfWorkers <= 0) – спроба конвертувати текст з елемента *txtNumberOfWorkers* у ціле число. Якщо конвертація не вдалася або введене число менше або дорівнює нулю, виводиться повідомлення про помилку та метод припиняє виконання.

lstResults.Items.Add();: Додає повідомлення про помилку до списку елементів *(ListBox)* у інтерфейсі користувача.

*string[] inputTasks = txtTasks.Text.Split(' ')*Розділяє текст з текстового поля *txtTasks* на масив рядків за пробілами. Кожен рядок представляє завдання у форматі "тип,тривалість".

```
CustomTask[] tasks = new CustomTask[inputTasks.Length];
for (int i = 0; i < inputTasks.Length; i++)
{
   string[] taskInfo = inputTasks[i].Split(',');
    if (taskInfo.Length == 2 && double.TryParse(taskInfo[1], out double taskValue) && taskValue >= 0)
    {
       tasks[i] = new CustomTask { Type = taskInfo[0], Time = taskValue };
    }
    else
       lstResults.Items.Add($);
```

Конвертує кожен рядок завдання в масив об'єктів *CustomTask* з валідацією.

Розбиває рядок завдання на дві частини – тип і тривалість, перевіряє коректність введення. Створює об'єкт *CustomTask* для кожного завдання та додає його до масиву *tasks*.

У випадку некоректного введення виводить повідомлення про помилку та припиняє виконання.

```
var workers = new List<CustomTask>[numberOfWorkers];
for (int i = 0; i < workers.Length; i++)
    workers[i] = new List<CustomTask>();
var orderedTasks = tasks.OrderByDescending(x => x.Time).ToArray();
double[] workerTotalTimes = new double[numberOfWorkers];
```

Створює масив списків для робітників *(workers)* та інші необхідні змінні.

Сортує завдання в порядку спадання тривалості.

```
foreach (CustomTask task in orderedTasks)
{
int assignedWorker = Array.IndexOf(workerTotalTimes, workerTotalTimes.Min());
 double startTime = Math.Max(workerTotalTimes[assignedWorker], START_TIME);
double endTime = startTime + task.Time;
workerTotalTimes[assignedWorker]=endTime;
workers[assignedWorker].Add(new CustomTask
{ Type = task.Type,
Time = task.Time,
StartTime = startTime,
EndTime = endTime });
 }
```

Цикл *foreach (CustomTask task in orderedTasks):*

Цей цикл пройдеться по кожному завданню (*task*) в порядку, встановленому змінною *orderedTasks*. Знаходження індексу робітника, який має найменший час в *workerTotalTimes*. Розрахунок часу початку *(startTime)* для завдання, забезпечуючи, що початок не раніше *START_TIME* (12:00).

Розрахунок часу завершення *(endTime)* для завдання.

Оновлення загального часу для призначеного робітника *(workerTotalTimes)*.

Збереження інформації про завдання для робітника в *workers*.

```
lstResults.Items.Clear(); for (int i = 0; i < workers.Length; i++)
{
 foreach (var task in workers[i])
{
lstResults.Items.Add($"Робітник {i + 1}:
{
GetTaskCode(task.Type)}({task.Time}) призначено Робітникові {i + 1}" + $" |
Початок: {ConvertMinutesToTime(task.StartTime)},
Кінець:{ConvertMinutesToTime(task.EndTime)}");
}
```

Очищення елементів списку *lstResults*.

Для кожного робітника виведення детальної інформації про призначені завдання.

double lastTaskEndTime = workers[i].Last().EndTime;

Знаходить час завершення останнього завдання для конкретного робітника.

workers[i]: Доступ до списку завдань, які були призначені робітнику з індексом *i*.

.Last(): Вибирає останнє завдання у списку.

EndTime: Отримує час завершення цього завдання.

lstResults.Items.Add($"Час завершення для Робітника {i + 1}: {Convert MinutesTTime(lastTaskEndTime)}");

Додає інформацію про час завершення останнього завдання робітника до списку *lstResults.*

lstResults.Items.Add(...): Додає новий елемент до списку *lstResults*.

*"*Час завершення для Робітника *{i + 1}: {Convert MinutesToTime(lastTaskEndTime)}":* Рядок, який включає інформацію про час завершення останнього завдання для конкретного робітника.

```
private static string GetTaskCode(string taskType)
{
    switch (taskType)
    {
        case "1":
            return "1-TO";
        case "2":
            return "2-refill";
        case "3":
            return "3-repair";
        case "4":
            return "4-diagnostics";
        default:
            return taskType;
    }
}
```

Повертає строковий код для вказаного типу завдання згідно з мапінгом типів.

switch (taskType): Визначає, який код повернути в залежності від значення *taskType.*

default: return taskType;: Повертає оригінальний тип, якщо його не знайдено в мапінгу.

```
private static string ConvertMinutesToTime(double minutes)
{
    int hours = (int)(minutes / 60);
    int mins = (int)(minutes % 60);
    return $"{hours:D2}:{mins:D2}";
}
```

Перетворює час у хвилинах в формат часу (години:хвилини).

int hours = (int)(minutes / 60): Розраховує кількість годин з хвилин.

int mins = (int)(minutes % 60): Розраховує кількість хвилин залишку від годин.

return $"{hours:D2}:{mins:D2}": Повертає форматований рядок у вигляді "години:хвилини".

```
public class CustomTask
{
    public string Type { get; set; }
    public double Time { get; set; }
    public double StartTime { get; set; }
    public double EndTime { get; set; }
}
```

Визначає клас *CustomTask*, який представляє завдання з властивостями *Type* (тип), *Time* (тривалість), *StartTime* (час початку) і *EndTime* (час завершення).

Після застосування всіх фільтрів, запит виконується за допомогою методу *.all()*, який повертає усі рейси, що відповідають заданим критеріям.

Результати запиту передаються для відображення користувачу.

Результатом виконання програми є відображення результатів в вікні програми, яка зображена на рис. 3.2.

```
Worker 1: 4-diagnostics(100); 1-TO(60); 4-diagnostics(40); 1-TO(20). Maxworkertime=220
4-diagnostics(100) assigned to Worker 1 | Start Time: 0, End Time: 100
  1-TO(60) assigned to Worker 1 | Start Time: 100, End Time: 160
  4-diagnostics(40) assigned to Worker 1 | Start Time: 160, End Time: 200
  1-TO(20) assigned to Worker 1 | Start Time: 200, End Time: 220
Worker 2: 4-diagnostics(100); 1-TO(60); 4-diagnostics(40); 3-repair(20). Maxworkertime=220
  4-diagnostics(100) assigned to Worker 2 | Start Time: 0, End Time: 100
  1-TO(60) assigned to Worker 2 | Start Time: 100, End Time: 160
  4-diagnostics(40) assigned to Worker 2 | Start Time: 160, End Time: 200
  3-repair(20) assigned to Worker 2 | Start Time: 200, End Time: 220
Worker 3: 3-repair(90); 2-refill(80); 2-refill(30); 1-TO(20). Maxworkertime=220
  3-repair(90) assigned to Worker 3 | Start Time: 0, End Time: 90
  2-refill(80) assigned to Worker 3 | Start Time: 90, End Time: 170
  2-refill(30) assigned to Worker 3 | Start Time: 170, End Time: 200
  1-TO(20) assigned to Worker 3 | Start Time: 200, End Time: 220
Worker 4: 3-repair(90); 2-refill(80); 2-refill(30); 3-repair(20). Maxworkertime=220
  3-repair(90) assigned to Worker 4 | Start Time: 0, End Time: 90
  2-refill(80) assigned to Worker 4 | Start Time: 90, End Time: 170
2-refill(30) assigned to Worker 4 | Start Time: 170, End Time: 200
  3-repair(20) assigned to Worker 4 | Start Time: 200, End Time: 220
Maximum service time: 220
```

Рис. 3.2. Вікно завершення програми

Загалом, для чіткого розуміння властивостей та взаємодії класів системи між собою в програмі, було створено діаграму класів системи, представлену на рис. 3.3.

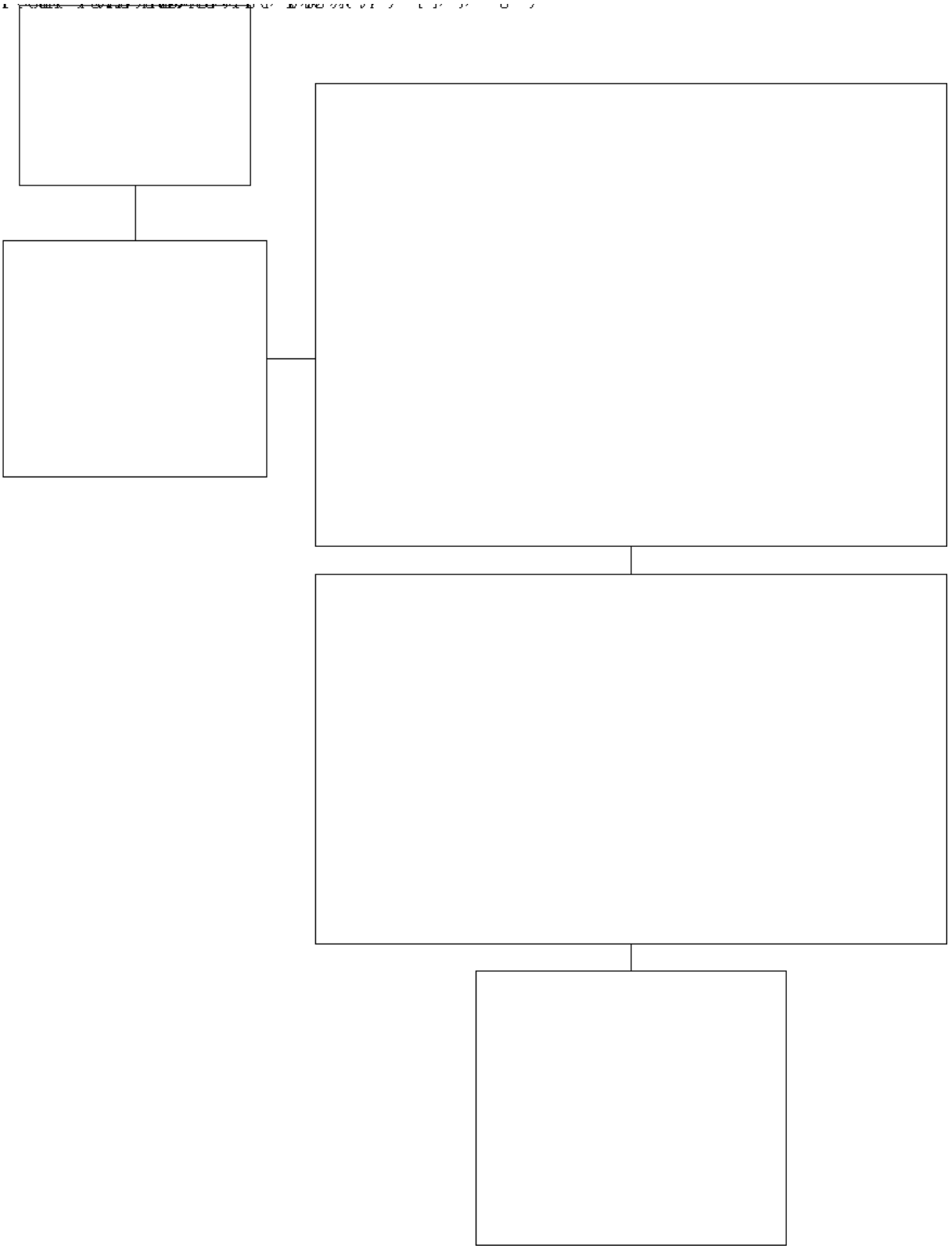

Рис. 3.3. Діаграма класів системи

3.3. Інтерфейс модуля

Існує безліч *GUI* бібліотек, проте для більшості *.NET* розробників найцікавішими понині залишаються *WinForms* та *WPF*. *Windows Presentation Foundation (WPF)* є фреймворком для розробки графічних інтерфейсів користувача для десктопних застосунків у середовищі *Windows*. Заснований на *.NET Framework*, *WPF* використовує декларативну мову розмітки *XAML* для визначення інтерфейсу та надає розширені можливості для створення відмінно виглядаючих та функціональних додатків. Цей фреймворк дозволяє розробникам легко створювати багатошарові додатки з різноманітними елементами управління, використовуючи гнучку систему стилів і шаблонів. *WPF* також надає механізм *Data Binding* для автоматизованої синхронізації даних між об'єктами додатка та його інтерфейсом.

Однією з ключових особливостей є підтримка векторної графіки, що дозволяє створювати гнучкі та масштабовані інтерфейси. Завдяки *XAML*, розробники можуть легко визначати структуру і зовнішній вигляд елементів інтерфейсу. *WPF* також підтримує різноманітні можливості анімації та переходів, дозволяючи створювати вражаючі ефекти та покращувати користувацький досвід. Модель програмування *MVVM* допомагає відділити логіку додатка від його інтерфейсу, полегшуючи тестування та підтримку коду.

Однією з найважливіших переваг *WPF* є його підтримка векторної графіки, що дозволяє створювати інтерфейси, які виглядають вражаюче на різних роздільних здатностях екранів і пристосовуються до розмірів вікна додатку. Це робить *WPF* ідеальним вибором для розробки додатків, які мають бути адаптивними та ефектними.

Модель програмування *MVVM (Model-View-ViewModel)* є ще однією ключовою особливістю *WPF*. Ця модель дозволяє відділити бізнес-логіку від представлення, що сприяє покращенню тестованості коду та управлінню складністю проектів.

WPF підтримує багатошаровість, де елементи інтерфейсу можуть бути розташовані один на іншому, що важливо для створення складних та

багатофункціональних додатків. Ця функція відкриває широкі можливості для розробників у створенні інтерфейсів з різноманітними елементами управління.

За допомогою *WPF* можна також легко застосовувати анімації та переходи, що додає інтерактивності та динаміки до додатків. Це робить інтерфейс користувача більш привабливим для користування.

Інтеграція *WPF* з іншими технологіями *.NET*, такими як *ASP.NET* та *Entity Framework*, розширює можливості розробників у створенні комплексних рішень, інтегрованих з різними платформами та службами.

Інструменти розробки, такі як *Visual Studio*, забезпечують продуктивне та зручне середовище для створення, налагодження та розгортання *WPF*-додатків. *Windows Presentation Foundation* визначається своєю гнучкістю, адаптивністю та можливістю створювати додатки, які не лише виглядають привабливо, але і мають високий рівень функціональності.

WPF підтримує можливість створення власних елементів управління та шаблонів, що дозволяє розробникам створювати унікальні та спеціалізовані інтерфейсні елементи для відображення інформації. Це важливо для тих, хто прагне надати своєму додатку відмінний вигляд та функціональність.

WPF також підтримує технологію подій, що дозволяє програмістам легко реагувати на події, такі як натискання кнопок чи зміна значень у текстових полях. Це робить інтерактивність додатків більш ефективною та користувацьки орієнтованою.

Інтегрований із іншими технологіями *.NET, WPF* робить можливим створення комплексних додатків, інтегрованих з іншими платформами та службами. Інструменти розробки, такі як *Visual Studio*, забезпечують зручне середовище для розробки, налагодження та розгортання *WPF*-додатків.

Саме завдяки цим перевагам інтерфейс програми було зроблено за допомогою *WPF* форм, з наступними рядками коду:

```
<Window x:Class="Idk.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="Планувальник завдань" Height="400" Width="600"
```

```
    <Grid>
        <StackPanel Margin="10">
            <TextBlock Text="Введіть кількість робітників:"/>
            <TextBox Name="txtNumberOfWorkers" Margin="0,0,0,10"/
                    <TextBlock Text="Введіть завдання у форматі 'тип1,тривалість1 тип2,тривалість2 ...'"/>
            <TextBox Name="txtTasks" Margin="0,0,0,10"/>
            <Button Content="Планувати завдання" Click="ScheduleTasks_Click"/>
            <ListBox Name="lstResults" Margin="0,10,0,0"/>
        </StackPanel>
      </Grid>
    </Window>
```

<Window> – Визначає головне вікно програми.

x:Class="Idk.MainWindow" – Зазначає, що код для вікна знаходиться в файлі *MainWindow.xaml.cs.*

xmlns – Задає XML-простір імен, який використовується в розмітці.

<Grid> – Вміщає всі елементи інтерфейсу у вигляді сітки.

StackPanel – Контейнер, що дозволяє розмістити елементи один над іншим.

TextBlock – Текстовий блок для введення кількості робітників.

TextBox – Поле для введення кількості робітників.

TextBlock – Текстовий блок для введення завдань.

TextBox – Поле для введення завдань.

Button – Кнопка для запуску планування завдань.

ListBox – Список для відображення результатів.

Вказані атрибути та властивості для кожного елементу призначені для забезпечення його відображення та взаємодії з користувачем відповідно. Ця розмітка визначає основний інтерфейс *WPF*-додатка, де користувач може вводити кількість робітників, завдання та планувати їх розподіл.

На першій сторінці слід ввести всі необхідні параметри та натиснути копку «Планувати завдання», вікно з введенням параметрів зображено на рис. 3.4.

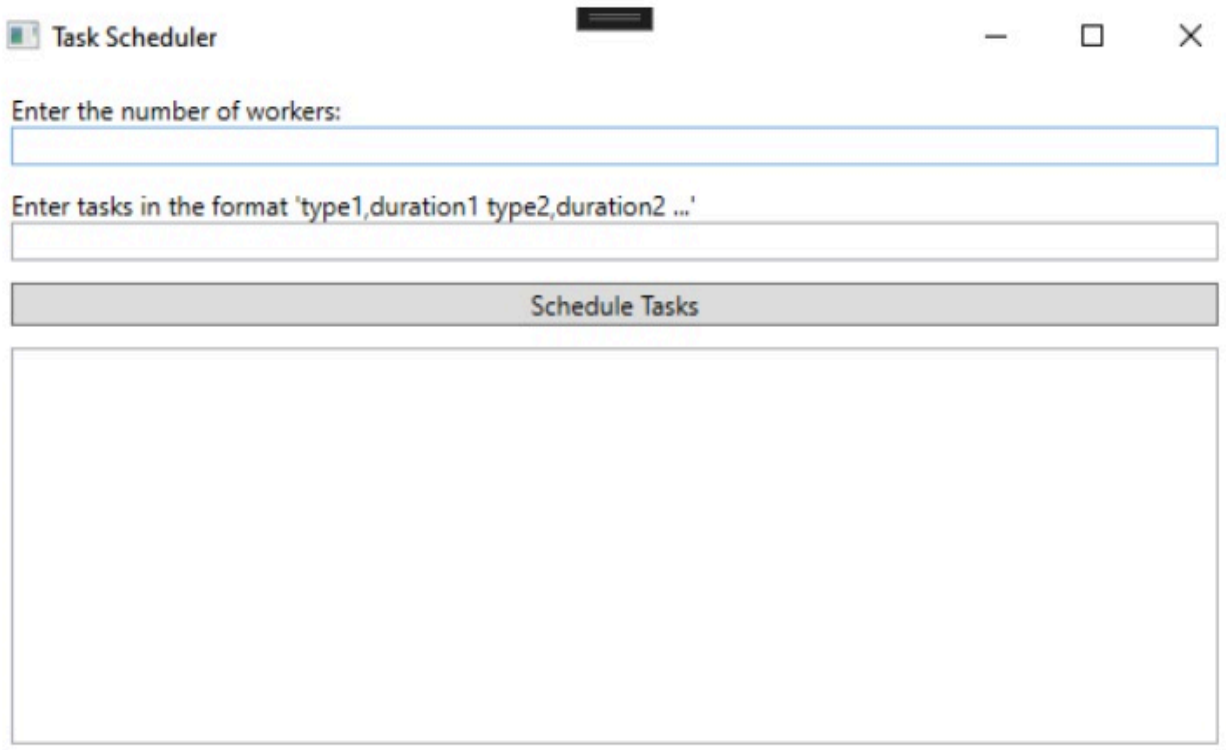

Рис. 3.4. Вікно з введенням параметрів

Після натискання кнопки планування модуль відсортує всі задачі, розподілить їх для роботи між виконавцями та виведе результат, що зображено на рис. 3.5.

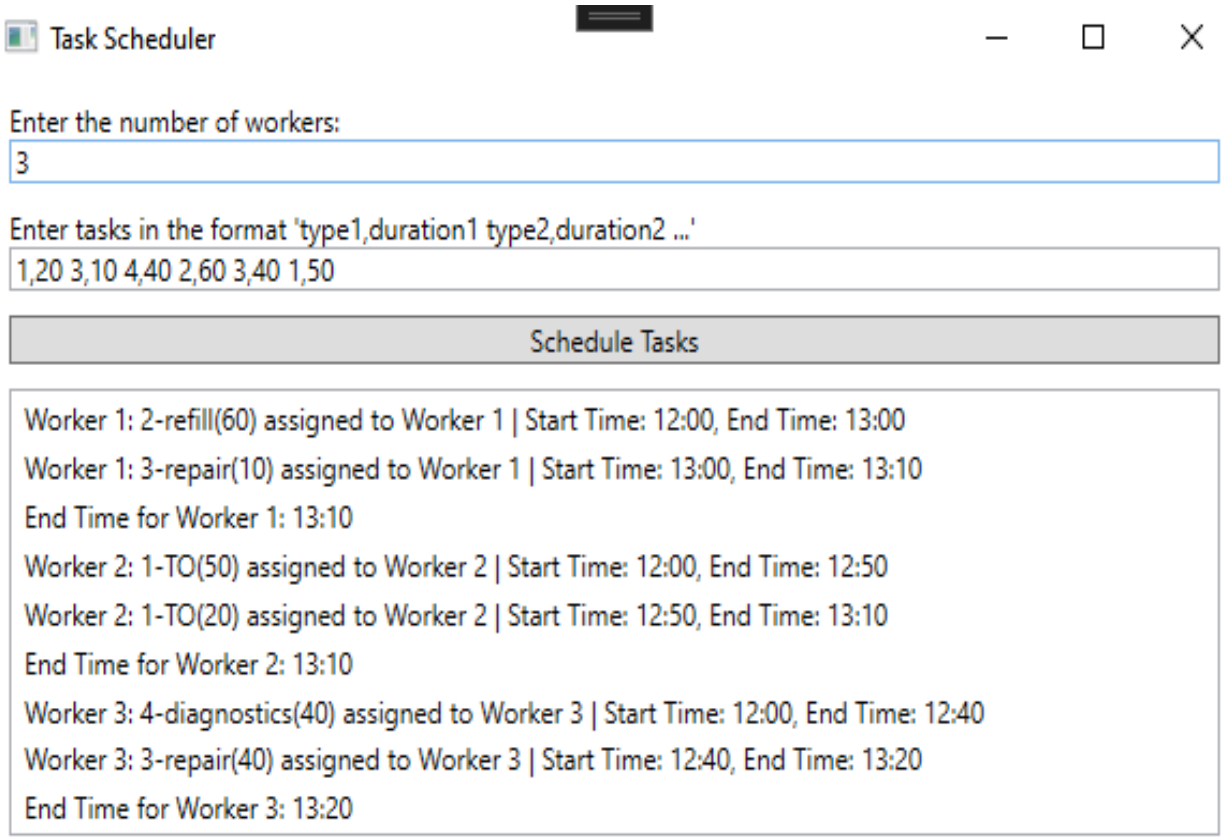

Рис. 3.5. Вікно результатів виконання програми.

Було проведено проєктування функціоналу системи і її внутрішньої будови, розроблено основні алгоритми програми та створено графічний інтерфейс. Окрім цього було проведено аналітичний огляд інструментальних засобів, які були використані під час розробки програмного продукту, а саме *WPF* для відображення інтерфейсу користувача.

3.4. Висновки до роділу

В третьому розділі було проведено фінальні дослідження та обгрунтовування. Обгрунтовано вибір мови програмування та фреймворку. Проаналізовано розробку системи планування обслуговування за допомогою евристичного алгоритму. В першу чергу було проведено огляд життєвого циклу і архітектури майбутнього програмного продукту. Було проведено дослідження функціоналу системи і її внутрішньої будови, розроблено основні алгоритми програми та створено графічний інтерфейс.

Завдяки своїй потужній та модульній архітектурі, а також зручному синтаксису, *C#* забезпечує високий рівень продуктивності у галузі обробки природної мови.

Мова програмування *C#* також використовується як мова сценаріїв у різноманітних програмних продуктах, включаючи *Abaqus, FreeCAD*, програми для обробки зображень та графічних редакторів. Її використання дозволяє розробникам ефективно створювати високоякісні додатки з відмінною продуктивністю та широким спектром можливостей.

Окрім цього було проведено аналітичний огляд інструментальних засобів, які були використані під час розробки програмного продукту, до яких входять *C#*, та фреймворк *.NET*. Попри всі значні переваги, *.NET* має незначні обмеження, тому був найкращим вибором для швидкості опрацювання вхідних даних.

Підсумовуючи, *C#* залишається одним з найкращих виборів створення програмного модулю через свою простоту, гнучкість, велику спільноту та багатий екосистему розширень. Це робить його ідеальним інструментом для широкого спектра проєктів, від простих персональних блогів до складних корпоративних систем.

ВИСНОВКИ

У процесі виконання кваліфікаційної роботи було створено модуль для планування технічного обслуговування технічних суден, що дозволяє ефективно знаходити, аналізувати та планувати рокзлад задач. Модуль відповідає сучасним тенденціям і вимогам в авіаційній галузі та є зручним інструментом для ремонтних бригад, аналітиків, аеропортів, забезпечуючи швидкий доступ до актуальної інформації, можливості швидкого прийняття рішення відповідно того чи іншого обєкту технічного обслуговування.

Було досліджено й проаналізовано саму технологію планування технічного обслуговування повітряних суден. Доведена можливість застосування модулю у вирішенні задач технічного обслуговування, розв’язання задач планування технічного обслуговування повітряних суден за допомогою жадібного евристичного алгоритму.

Було здійснено аналіз існуючих методів планування технічного обслуговування повітряних суден. Було зважено переваги і недоліки методів планування технічного обслуговування повітряних суден. Визначено один із найоптимальніших методів планування технічного обслуговування повітряних суден, а саме метод виначення найбільшого пріоритету.

В контексті планування технічного обслуговування повітряних суден, евристичний метод може бути застосований для складання розкладу виконання робіт за допомогою інформації про доступні ресурси, розклади польотів та інші параметри. Однією з головних переваг евристичного методу в порівнянні з традиційними методами є його швидкість та можливість обробки великих обсягів даних. Евристика дозволяє знайти наближене рішення за короткий проміжок часу та аналізувати багатофакторні залежності між різними параметрами. Чітко зазначено, що за допомогою евристичного методу можна швидко визначити оптимальний час для проведення технічного обслуговування, враховуючи наявні ресурси та міжнародні розклади польотів.

Евристичний метод може пропонувати розв'язання проблем, які не завжди зручно формалізувати. У випадку непередбачених змін у розкладі польотів або вмикання тимчасового резервного повітряного судна, евристичний метод може прискорити планування та забезпечити швидку адаптацію розкладу виконання робіт. При застосуванні евристичного методу до планування технічного обслуговування повітряних суден, необхідно бути уважним до граничних значень та збалансовувати різні фактори при прийнятті рішення. Можливість застосування евристичного методу до планування технічного обслуговування повітряних суден пропонує багато переваг в порівнянні з традиційними методами. Швидкість обробки даних, здатність до адаптації та прийняття рішень, а також урахування непередбачених змін є головними перевагами евристичного методу. Незважаючи на це, застосування евристичного методу до планування технічного обслуговування повітряних суден має значний потенціал.

Розробка модулю була здійснена з використанням мови програмування *C#* та фреймворка *.Net* що дозволило інтегрувати потужні інструменти для роботи з даними та створити інтуїтивно зрозумілий інтерфейс. Інтерфейс в свою чергу був обраний *WPF*, який є частиною технології *.NET Framework* і призначений для створення *Windows*-додатків з багатошаровим графічним інтерфейсом.

Додаток має значний потенціал для подальшого розвитку та вдосконалення. Можливості для розширення функціоналу включають інтеграцію з додатковими даними, розширення бази даних, персоналізації пропозицій для користувачів. Пропонований модуль може бути використаний для розробки інформаційних систем для обслуговування повітряних суден. Можуть бути застосовані при розробці систем, схожих за вимогами функціональності та надійності.

Пропонований модуль може бути використаний у складі систем графіків планування технічного обслуговування суден, графіків планування заправлення суден, може бути використаний у складі систем технічного обслуговування та управління повітряними суднами для впровадження сучасних підходів до автоматизації та оптимізації процесів. Завдяки своїй простоті він легко може інтегруватися з іншими компонентами авіаційних інформаційних систем, такими як

системи моніторингу, датчиків стану, та систем управління обладнанням. Модуль також може підтримувати звітність та аналіз результатів обслуговування для поліпшення стратегій технічного обслуговування у майбутньому. Його впровадження може сприяти підвищенню оперативної ефективності та забезпечити високий рівень надійності.

Таким чином, розроблений програмний модуль є актуальним та перспективним інструментом для різних категорій користувачів, який сприяє зростанню інформованості та прозорості у сфері авіаційних послуг. Результати проєкту підтверджують його важливість та практичну цінність, а розроблена система може бути рекомендована до широкого впровадження та використання в галузі технічного обслуговування.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Бойченко С.В., Іванченко О.В. Положення про дипломні роботи (проекти) випускників Національного авіаційного університету. – К.: НАУ 2017. – 63 с.
2. ДСТУ 3008-95. Документація Звіти у сфері науки і техніки. Структура і правила оормлення. – 39 с.
3. ДСТУ ГОСТ 7.1:2006 «Бібліографічний запис. Бібліографічний опис. Загальні вимоги та правила складання».
4. ДСТУ 8302:2015 «Інформація та документація. Бібліографічне посилання. Загальні положення та правила складання».
5. *Gormley C., Tong Z. Mastering Elasticsearch: Building a Scalable, Open Source Search Engine. O'Reilly Media,* 2013, 354 *p.*
6. *Grossman D., Frieder O., Karlgren J. Information Retrieval: Algorithms and Heuristics. Springer,* 2004, 292 *p.*
7. *Papadimitriou, C. H., & Steiglitz, K. Combinatorial Optimization: Algorithms and Complexity. Dover Publications,* 1998, 200 *p.*
8. *Blum, C., & Roli, A. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. ACM Computing Surveys (CSUR)* . 2003, 268-308 *p.*
9. *Taha, H. A. Operations Research: An Introduction. Pearson Education,* 2011, 120 *p.*
10. *Cotta, C., & Moscato, P. A gentle introduction to memetic algorithms. In Handbook of Metaheuristics . Springer. 2*003, 105-144 *p.*
11. *Anderson, D. P. BOINC: A system for public-resource computing and storage. In Grid Computing,* 2004, 4-10 *p.*
12. *Zitzler, E., Laumanns, M., & Thiele, L. SPEA2: Improving the strength Pareto evolutionary algorithm. TIK-report,* 2002, 103 *p.*
13. *Glover, F., & Kochenberger, G. A. Handbook of Metaheuristics (International Series in Operations Research & Management Science). Springer,* 2003 , 20 *p.*
14. *Dagum, L., & Menon, R. OpenMP: An industry standard API for shared-memory programming. IEEE Computational Science & Engineering, 1998, 46-55 p.*

15. *Taillard, É. D. Benchmarks for basic scheduling problems. European Journal of Operational Research),* 1999, 278-285 *p.*
16. *Braekers, K., Ramaekers, K., & Caris, A. A hybrid evolutionary algorithm for the periodic vehicle routing problem with time windows. Computers & Operations Research,* 2016, 53-62 *p.*
17. *Law, A. M., & Kelton, W. D. Simulation Modeling and Analysis. McGraw-Hill. 2014.*
18. *Osman, I. H. Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. Annals of Operations Research,* 2003, 421-451 *p.*
19. *Barnhart, C., Johnson, E. L., Nemhauser, G., Savelsbergh, M., & Vance, P. H.. Branch-and-price: Column generation for solving huge integer programs. Operations Research, 2010*, 316-329 *p.*
20. *Bierwirth, C., & Mattfeld, D. C. A parallel genetic algorithm for the resource-constrained project scheduling problem.* 2014, 634-644 *p.*
21. *Salhi, S., & Nagy, G. . A cluster insertion heuristic for the VRP. Computers & Operations Research,* 2007, 1025-1037 *p.*
22. *Toth, P., & Vigo, D. (Eds.).The Vehicle Routing Problem (Monographs on Discrete Mathematics and Applications). Society for Industrial and Applied Mathematics.* 1987, 386 *p.*
23. *Banzhaf, W., Nordin, P., Keller, R. E., & Francone, F. D. Genetic Programming: An Introduction. Morgan Kaufmann.* 1998, 200 *p.*
24. *Lawler, E. L. . A Combinatorial Optimization Algorithm for Solving the Set-Covering Problem. Management Science,* 1977, 323-335 *p.*
25. *Drexl, M., & Kimms, A. . Metaheuristics for Production Scheduling. Springer.* 2014, 40 *p.*
26. *Glover, F.. Tabu Search—Part I. ORSA Journal on Computing,* 1989, 190-206 *p.*
27. *Bergh, F., & Engelbrecht, A. P. A Study of Particle Swarm Optimization Particle Trajectories. Information Sciences,* 2004, 937-971 *p.*
28. *Kochenderfer, M. J., & Wheeler, T. A. Algorithms for Optimization. MIT Press.* 2006, 90 *p.*

29. *Blum, C. . Ant colony optimization: Introduction and recent trends. Physics of Life Reviews,* 2005, 353-373 *p.*

30. *Yang, X.-S. A New Metaheuristic Bat-Inspired Algorithm. Studies in Computational Intelligence,* 2010, 654 p.

31. *Kennedy, J., & Eberhart, R. Particle Swarm Optimization. In Proceedings of ICNN'95 - International Conference on Neural Networks,* 1995, 45*0 p.*

32. *Mirjalili, S., & Lewis, A. The Whale Optimization Algorithm. Advances in Engineering Software*, 2016, 51-67.

33. *Storn, R., & Price, K.. Differential Evolution – A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces. Journal of Global Optimization, 1997, 341-359 p.*