МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВІДОКРЕМЛЕНИЙ СТРУКТУРНИЙ ПІДРОЗДІЛ
«ФАХОВИЙ КОЛЕДЖ ІНЖЕНЕРІЇ ТА УПРАВЛІННЯ
НАЦІОНАЛЬНОГО АВІАЦІЙНОГО УНІВЕРСИТЕТУ»

ДОПУСТИТИ ДО ЗАХИСТУ

Заступник директора з НР

_____ О.В. Родіонова

«____»_____ 2021 р.

КВАЛІФІКАЦІЙНА РОБОТА

ЗДОБУВАЧА ОСВІТНЬОГО СТУПЕНЯ
«БАКАЛАВР»

Тема: «Програма оцінки забруднення водойм» _____

Автор: Іванов Михайло Віталійович

Керівник проєкту: О.В. Пономаренко

Нормконтролер: В.М. Кругляк

Київ 2021

ВІДОКРЕМЛЕНИЙ СТРУКТУРНИЙ ПІДРОЗДІЛ
«ФАХОВИЙ КОЛЕДЖ ІНЖЕНЕРІЇ ТА УПРАВЛІННЯ
НАЦІОНАЛЬНОГО АВІАЦІЙНОГО УНІВЕРСИТЕТУ»

Циклова комісія: Інженері програмного забезпечення

Освітнього ступеня: «Бакалавр»
Спеціальність: 123 Комп'ютерна інженерія
Освітньо-професійна програма: «Комп'ютерна інженерія»

ЗАТВЕРДЖУЮ

Голова циклової комісії

_____ Н.А. Рябчук

«____»_____ 2021 р.

ЗАВДАННЯ

на виконання дипломного проєкту

Іванова Михайла Віталійовича

1. Тема роботи: «Програма оцінки забруднення водойм»

затверджена наказом від «15» березня 2021 року № 28-Ст.

2. Термін виконання: з 12.04.2021р. по 20.06.2021р.

3. Вихідні дані: Розробити програму оцінки забруднення водойм яка буде зображати зони з забрудненням водойми та якісно їх оцінювати, Програма має мати зручний та зрозумілий користувацький інтерфейс.

4. Зміст пояснювальної записки:

У 1 розділі проаналізувати завдання на проєкт та основні методи його розв'язання.

У 2 розділі наведести опис основних етапів розробки.

У 3 розділі описати основні вимоги охорони праці.

КАЛЕНДАРНИЙ ПЛАН

виконання дипломної роботи

№ п/п	Етапи виконання кваліфікаційної роботи	Термін виконання етапів	Примітка
1.	Уточнення постановки задачі	12.04.2021	Виконано
2.	Аналіз літературних джерел	15.04.2021	Виконано
3.	Обґрунтування рішення	21.04.2021	Виконано
4.	Збір інформації	01.05.2021	Виконано
5.	Аналіз існуючих методів. Обгрунтування вибору мови програмування	06.05.2021	Виконано
6.	Виконання проєкту	08.06.2021	Виконано
7.	Оформлення і друк пояснювальної записки	15.06.2021	Виконано
8.	Оформлення презентації	17.06.2021	Виконано
9.	Отримання рецензій	20.06.2021	Виконано
10.	Захист проєкту		

Дипломник _____ Іванов М. В.
(підпис, дата) (П.І.Б.)

Дипломний керівник _____ Пономаренко О.В.
(підпис, дата) (П.І.Б.)

Консультант з охорони праці _____ Пешков І.В.
(підпис, дата) (П.І.Б.)

Зміст

Вступ

Мета кваліфікаційної роботи створити програмний модуль для аналізу поверхні водоймищ на предмет їх забруднення, з ціллю урегулювання екологічного стану планети, насамперед водоймищ.

Модуль призначений для полегшення аналізу зібраного маткерілу за допомогою безпілотних літальних апаратів або супутників, шляхом порівняння різних ділянок водної поверхні між собою та іншими еталонними фрагментами та демонстрації результатів у зрозумілому для користувача вигляді.

Актуальність данного програмного збезпечення пояснюється погіршенням екологічної ситуації, а саме забруднення води шляхом потрапляння до водоймищ неорганічного сміття природного походження, такого як: пісок, глина, бруд, неорганічного сміття неприроднього походження, такого як: папір, поліетилен, недопалки, зламаний інструмент та інших стороннії предметів та рідин, так і органічного у вигляді опавшого листя, зламаних гілок та заростлів.

Для вирішення поставленої задачі було виконано аналіз програм аналогів, які вже існують на ринку, проектування програмного забезпечення та розобка, аналіз засобів збору інформації та створення необхідних супутніх компонентів. Також було розроблено інструкцію для користувача програмним забезпеченням.

Під час розробки пз було проведено аналіз потенційного апаратного забезпечення необхідного для коректної роботи розроблюваного програмного забезпечення. Для аналізу апаратного забезпечення необхідно встановити параметри необхідні для якісної роботи розроблюваного програмного модуля, а саме: критерії для опрацьовуємого матеріалу, апаратні вимоги для роботи програмного забезпечення та апаратні вимоги до технічного засобу збору матеріалу для аналізу. Висновком такого аналізу стали мінімально необхідні потужності для обробки інформації, мінімально необхідні потужності до засобу збору інформації та мінімально необхідні парамети збираємої інформації з врахуванням фінансових затрат на необхідні компоненнти, та витрат на збір інформації та оперативність її обробки. Даний аналіз було проведено з метою підвищення доступності

програмного забезпечення, доступності його використання та зменшення затрат на проектування.

Сумуючи вище сказане було отримане доступне якісне та сучасне ПЗ що здатне працювати на відносно низькопотужному апаратному забезпеченні використовуючи для збору інформації середньопотужне апаратне забезпечення що дозволить виконати аналіз та залишитись в задовільних межах похибки.

1 Загальна частина

1.1 Постановка задачі

Мета створення програми для аналізу забруднення водної поверхні – створити програмне забезпечення яке допоможе оперативно аналізувати наявність забруднення водоймища, відмічати місця забруднення або підозрілі місця на які слід звернути увагу, проводити певні розрахунки такі як ступінь забруднення, відносна площа забруднення.

Програма призначена для полегшення виявлення забруднення водойми, визначення його ступеню, дати кількісну оцінку на основі якої розпочати комплекс дій що сприятимуть очищенню водойми.

Актуальність даного програмного забезпечення (пз) пояснюється підвищеним рівнем забрудення водоймищ України, що на сам перед впливає на якість грунтів, які в свою чергу на якість вирощуваних на них продуктів харчування і як наслідок здоров’я населення що їх вживає.

Відповідно до звітів експертної групи ООН від вживання неякісної води щороку помирає приблизно стільки ж людей, скільки й з-за злочинів, пов’язаних з насильством. Щорічно ця проблема забирає життя близько 1.8 мільйона дітей віком до 5 років. Ось до чого призводить забруднення води. У побуті ця проблема частково вирішується застосуванням фільтрів з активованим вугіллям, оснащених системою механічного очищення, а також здатних видаляти марганець й залізо, тим самим усуваючи наслідки забруднення води. На жаль, далеко не всі споживачі можуть дозволити собі покупку подібного обладнання. Крім того, у багатьох країнах комунальні служби нехтують необхідністю використання дезінфікуючих й коагулуючих хімічних реагентів.

Переважна більшість промислових й виробничих об’єктів в нашій країні не оснащена водоочисними спорудами взагалі. Підприємства просто виводять свої стоки в річки, без будь-якої фільтрації. Володіючи компанією, діяльність якої призводить до утворення подібних відходів, будь-яка поважаюча себе людина зобов’язана подбати про те, щоб її підприємство завдавало якомога менше шкоди

навколишньому середовищу. Адже відповідальність за екологічні наслідки забруднення гідросфери цілком й повністю лежить на плечах монополістів.

Крім побутових й промислових стічних вод в річки та озера країни регулярно потрапляють різні сільськогосподарські хімікати, 80% з яких практично не проходили ніяких лабораторних випробувань, з чого слідує непередбачуваність наслідків їх застосування.

Враховуючи вище сказане пожна підвести проміжні підсумки а саме: програмне забезпечення повинне працювати на переносних персональних комп'ютерах з високою автономністю високою продуктивністю зі зниженим енергоживленням таких як «ноутбук», що обумовлено необхідністю працювати в умовах «польових» робот. Також необхідно врахувати той момент що використовувати програмне забезпечення може користувач з базовим рівнем володіння персональним комп'ютером, для розуміння результатів виконання програми для аналізу забруднення водоймищ оператору програмним забезпеченням необідно мати повне уявлення про досліджувану водойму, її глубину та розташування, враховувати умови за яких проводився збір досліджуваного матеріалу у вигляді фото,

Мінімальні системні вимоги для коректної роботи програми:

- оперативна пам'ять: 2048 МБ;
- частота процесора: 2000 МГц;
- вільне місце на диску: 512 МБ;
- операційна система: Windows 8.

Рекомендовані системні вимоги для коректної роботи програми:

- оперативна пам'ять: 4096 МБ;
- частота процесора: 3000 МГц;
- вільне місце на диску: 512 МБ;
- операційна система: Windows 10.

1.2 Теоретичні відомості

Сучасні технології проектування автоматизованих систем (АС) надають потужні інструменти їх реалізації. Однак можуть виникати ситуації, коли технічно ідеально спроектована система не виправдовує очікувань і дуже часто це відбувається через недостатнє врахування людського фактору в процесі проектування.

Невід'ємною частиною будь-якого програмного продукту, який призначений для використання в інтерактивному режимі, є інтерфейс користувача (ІК). Він об'єднує в собі всі елементи і компоненти програми, які здатні впливати на взаємодію користувача з програмним забезпеченням.

Елементи користувацького інтерфейсу:

- засоби відображення інформації, пристрої та технології введення даних;
- відображувана інформація, формати і коди;
- командні режими, мова користувач-інтерфейс;
- сценарії діалогу і самі діалоги;
- зворотний зв'язок з користувачем;
- підтримка прийняття рішень у конкретній предметній області;
- порядок використання програми і документація на неї.

Проектування ІК перетворилося на самостійну проблему, яка найчастіше перевершує по складності проблему розробки кодів програми, і вимагає, як і процес проектування будь-якої складної системи, відповідних методів, засобів, і, природно, зусиль кваліфікованих фахівців. Саме застосування терміну ЛКІ являє собою спробу розробників програмного забезпечення відокремити, принаймні концептуально, функціональне призначення програмних продуктів від проблем, пов'язаних з організацією взаємодії користувача з цими продуктами. Такий поділ є необхідною умовою створення «дружніх» інтерфейсів по відношенню до користувачів програмних продуктів. Зауважимо, що саме поняття дружності - це вектор, який містить відповідно до міжнародної класифікації сім вимог до ІК:

- відповідність завдань, що вирішуються користувачем;

- легкість застосування;

- керованість;

- відповідність очікуванням користувача;

- стійкість до помилок;

- адаптованість / індивідуалізоване;

- легкість вивчення.

Сучасні підходи до проектування ІК базуються на визначеній методологічній основі, яка виділяє три ключові проблеми організації процесу проектування ІК:

- ідентифікація інформації, необхідної для проектування;

- визначення і структурування власне процесу проектування;

- цілі та порядок проведення експертизи ефективності ІК.

Узагальнена структура інформації для проектування інтерфейсу АС:

1. Оператор:

- ідентифікація;

- пріоритетність;

- сфери інтересів;

- очікування;

- вік, стать;

- антропометрія, відхилення;

- культура;

- мова (яка може залежати від завдання);

- ідентифікація груп;

- фактори стресу;

- асоціативні та ролеві моделі;

- освіта;

- рівень знання комп'ютера.

2. Організація праці:

- стратегія інформаційної системи;

- робочий процес;

- ідентифікація завдання;

- атрибути завдання;
- процедури, методи;
- вимоги інформації;
- відповідальність;
- процес рішення;
- рівень автоматизації;
- розпорядок роботи;
- робоче навантаження;
- гарантії;
- заробітна платня, стимули, вигоди;
- заохочення;
- набір і навчання;
- активність профоб'єднань.

3. Фізична середа:
- основна робоча середа (макросередовище);
- робоче місце (мікросередовище);
- обладнання, інструментальні засоби тощо;
- підтримка та експлуатація;
- освітленість;
- клімат;
- шум;
- вібрація;
- випромінювання;
- токсичні речовини.

Більшість методів проектування, що реалізують ту чи іншу методологію, сьогодні тяжіє до технологічних рішень та інструментальних засобів, і здебільшого орієнтовано на реалізацію технічних рішень. При цьому майже не приділяється уваги розгляду таких важливих завдань, пов'язаних з проектуванням, як збір та аналіз інформації, синтез і оцінка проектних рішень на ранніх етапах проектування.

А між тим, інструментальні програмні засоби є лише інструментом і самі по собі не гарантують правильних проектних рішень.

Інтерфейс за допомогою інструментальних засобів «збирається» з готових елементів – «кубиків». Однак з одних і тих же кубиків можна «побудувати» інтерфейси різної якості та з різними ергономічними властивостями. Тут багато що залежить від проектувальників, їх знань, досвіду і кваліфікації в області організації людино-комп'ютерної взаємодії.

Хто може проектувати ІК? Дуже часто подібним «конструюванням» доводиться займатися програмістам, рідше прикладним фахівцям, наприклад, технологам. Насправді, ж рішення цих проблем відноситься до сфери компетенції фахівців з людино-комп'ютерних інтерфейсів, або, як кажуть за кордоном, фахівцям в області USABILITY. Зараз вже в більшості посібників з розробки ІК, що включають компоненти візуального програмування вказується, що розробкою ІК повинні займатися спеціально підготовлені співробітники і категорично не рекомендується залучати для цієї мети виключно програмістів.

При цьому можна відзначити одну цікаву особливість. Більшість фахівців, особливо програмістів, які досить довго просиділи за екранами моніторів, вважають себе досвідченими цінувальниками та знавцями людино-комп'ютерних інтерфейсів. Переконати їх у зворотному буває вкрай важко, хоча по суті вони є користувачами, правда, дуже кваліфікованими.

Висновок очевидний - залучення консультантів зі сторони. Їх небагато, але вони є. Підхід найбільш доцільний, коли робота разова. Якщо ж фірма реалізує безліч проектів або в рамках одного проекту передбачається тривала еволюція системи, то бажано мати (готувати) своїх фахівців.

Методологія створення ІК існує в зародковому стані. Програміст змушений перетворюватися на художника і психолога одночасно, тому потрібно адекватне розуміння потреб людини. Авторами найбільш вдалих інтерфейсів є фахівці одночасно в області обчислювальної техніки, психології та дизайну.

Хто ж може проектувати ІК? У цій комплексній роботі повинні брати участь:

- постановник задач, який досконально знає вирішувані завдання;

- інженерний психолог (ергономіст), що враховує особливості психофізіології людини;

- програміст, який створює додаток;

- дизайнер, який робить інтерфейс привабливим.

Природно, в одній людині, такі різнорідні знання практично не зустрічаються. Отже - бригада. Основним, звичайно ж, є постановник завдань. Кращим постановником є людина, яка вміє ефективно спілкуватися з Замовником.

Нормативно - технічна база – стандарти ІК. За кордоном існує ціла індустрія проектування ІК, в будь-якій фірмі, що займається розробкою програмного забезпечення існують спеціалізовані ергономічні служби або відділи, є величезна кількість методичних посібників та, що особливо важливо, велике число нормативно-технічної документації, в тому числі на рівні міжнародних стандартів. У СНД ж ситуація зовсім інша: мізерно мало кваліфікованих спеціалістів, практично немає довідкової літератури, не застосовуються професійні стандарти.

Ситуація ускладнюється ще й тим, що інтерфейси більшості інструментальних систем, з якими доводиться мати справу розробникам програмного забезпечення, відносяться до класу так званих офісних інтерфейсів, які беруть свій початок від офісних систем. Інтерфейси же програм АСНД (АС наукових досліджень) і АСУТП (АС управління технологічним процесом), наприклад, відносяться до класу інформаційно-керуючих, ергономіка яких кардинально відрізняється від офісних і, як правило, абсолютно незнайома нашим розробникам.

У тих випадках, коли фірма реалізує безліч проектів необхідно підготувати також одне або декілька корпоративних керівництв з проектування ІК. Цей підхід достатньо ефективний і не пов'язаний зі значними витратами, тому має сенс зупинитися на ньому докладніше.

Стандартизація є одним з найбільш доступних інструментів якісних інтерфейсів. Сучасна концепція стандартизації людино-комп'ютерних інтерфейсів базується на ієрархії концепції:

1. Проектування користувацького інтерфейсу.

2. Програмна реалізація інтерфейсу.

3. Стилі інтерфейсів.

Перший рівень ієрархії визначає основні методи та інструменти проектування графічного інтерфейсу, другий - особливості його програмної реалізації, зокрема, особливості реалізації та сумісності для різних комп'ютерних платформ, а третій - особливості реалізації для різних класів систем. В цілому розробка і дотримання ІК стандартів дозволяють забезпечити:

- високу продуктивність роботи користувачів, тому що користувачі отримають і будуть використовувати легкосприйнятні засоби вирішення їх професійних завдань з мінімальним ризиком зустріти труднощі або перешкоди;

- малий час навчання - користувачу буде достатньо вивчити одну систему і отримані знання та навички стануть базовими при використанні інших систем;

- скорочення часу розробки - не буде необхідності проектувати кожну компоненту окремо, оскільки з'являться попередньо розроблені відповідно до нормативів універсальні програмні компоненти. Використання цих компонент в поєднанні з керівництвами по людино-комп'ютерним інтерфейсам, формам, що деталізуються, застосування цих компонент на рівні конкретних типів додатків, дозволить мінімізувати відмінності інтерфейсів і буде сприяти скороченню часу розробки.

Стандарти можуть мати різну форму і можуть одночасно служити цілям:

- керівництва є джерелами проектних рішень для проектувальників у частині розробки форматів відображуваної інформації та інтерактивних процедур;

- керівництва забезпечують загальний підхід до систематизованого проектування на основі фундаментальних принципів ергономічного проектування;

- керівництва сприяють розширенню рамок критеріїв персонального вибору та зменшення вимог до підготовки та якостей операторів всіх систем.

Для досягнення цих цілей в процесі розробки керівництв ставляться наступні завдання:

- ідентифікувати та встановити всі функції і завдання, які вирішуються системою і операційним середовищем. Це сприяє розумінню загальної динаміки систем;

- виконати аналіз можливостей і обмежень користувачів системи. Це дозволить краще розподілити функції між людиною і машиною і гарантує виконання приписаних функцій з боку людини;

- систематизувати правила проектування інтерфейсу.

Стилі інтерфейсу. Зауважимо, що корпоративні керівництва з проектування ІК мають всі без винятку відомі фірми, що займаються програмуванням. Керівництва деяких фірм стали основою відповідного ергономічного стилю інтерфейсу. Прикладами найбільш часто використовуваних комерційних стилів є:

- OSF/Motif;
- SUN/OpenLook;
- Microsoft Windows;
- Apple.

Наші розробники з ряду причин найкраще знайомі зі стилем Microsoft Windows. Цей стиль найкраще пристосований до офісних додатків і менш за все - до діяльності типу оперативно-тактичного управління.

Між всіма стилями існують відмінності, які можуть бути згруповані в наступні три досить широкі категорії:

- термінологія - це розходження в назвах, що присвоюються та описують функції та елементи. Основна відмінність полягає у використанні різних назв і формулювань для опису еквівалентних або подібних функцій та елементів. Але іноді одна й та ж назва використовується для абсолютно різних елементів. Прикладом різних назв для подібних елементів: Motif використовує термін «радіокнопка», а OpenLook – «виключний перемикач».

- зовнішність - це розходження в графічному поданні;

- сприйняття - це розходження в діях, які повинен вжити користувач, взаємодіючи з додатком. Відмінності пов'язані із застосуванням і використанням

кнопок миші, функціональних клавіш, мнемоніки і прискорювачів, деяких підходів по керуванню.

Вимоги до ІК, принципи реалізації інтерфейсу.

Мінімізація зусиль користувача при виконанні роботи:

- скорочення тривалості операцій читання, редагування і пошуку інформації;

- зменшення часу навігації і вибору команди;

- підвищення загальної продуктивності користувача, що полягає в обсязі оброблених даних за певний період часу;

- збільшення тривалості стійкої роботи користувача тощо.

Стильова гнучкість - можливість використовувати різні інтерфейси з одним і тим же додатком, на практиці реалізується за допомогою таблиці стилів, в тому числі можливості вибору користувачем власних установок ІК (колір, ікони, підказки тощо).

Нарощування функціональності - можливість розвивати додаток без руйнування (тобто залишаючись в рамках) існуючого інтерфейсу.

Масштабованість - можливість легко налаштовувати і розширювати як інтерфейс, так і сам додаток при збільшенні кількості користувачів, робочих місць, обсягу і характеристик даних.

Адаптивність до дій користувача - додаток повинен допускати можливість введення даних і команд множиною різних способів (клавіатура, миша, інші пристрої) і багато варіативністю доступу до прикладних функцій (ікони, «гарячі клавіші», пункти меню), крім того, програма повинна враховувати можливість переходу і повернення від вікна до вікна, від режиму до режиму, і правильно обробляти такі ситуації.

Незалежність у ресурсах - для створення користувацького інтерфейсу повинні надаватися окремі ресурси, спрямовані на зберігання і обробку даних, необхідних для підтримки користувача (словники користувача, контекстно-залежні списки, набори даних за замовчуванням або за останнім запитом, історії запитів тощо).

Переносимість - при переході на іншу апаратну (програмну) платформу, повинно здійснюватися автоматичне перенесення і користувацького інтерфейсу, і кінцевого додатка.

Розробка ІК ведеться паралельно розробці програмного продукту в цілому і в основному передує його впровадженню. Процес розробки ергономічного ІК розбивається на наступні етапи:

1. Аналіз виробничої діяльності користувача, визначення і специфікація його бізнес-функцій. Формулювання вимог до роботи користувача. Побудова користувацької моделі даних (ERD), формування робочих місць.

2. Проектування ІК - вибір показників оцінки користувацького інтерфейсу. Розробка узагальненого сценарію взаємодії користувача з системою (функціональної моделі) і його попередня оцінка користувачами та Замовником (паперовий прототип ІК). Коригування і деталізація сценарію взаємодії, вибір і доповнення стандарту (керівництва) для побудови прототипу. Розробка макетів і прототипів ІК та їх оцінка у діловій грі, вибір остаточного варіанта. При проектуванні користувацького інтерфейсу наведена вище послідовність не є строго обов'язковою. Проектувальник може уявити діалог в екранних формах. Однак на цьому етапі головне погодити та затвердити не вид екрану (це вторинне і відображає швидше смак і майстерність розробника), а саме:

- структури і домени даних, що вводяться, коректуються і видаляються, дані NOT NULL;

- дії здійснювані при цьому користувачем, ІС, СКБД і при необхідності операційною системою;

- умови і шляхи переходу;

- умови підтримання цілісності та бізнес-правил;

- тригера;

- збережені процедури;

- навчання та/або функції допомоги користувачу;

Найбільш поширеною помилкою розробника є саме відсутність чіткого опрацювання виконуваних дій. Без цього подальша реалізація виявляється

неузгодженою і може виявитися не відповідної кваліфікаційним вимогам, а на практиці вимогам користувача.

3. Реалізація ІК в коді, створення тестової версії (візуалізація). Розробка засобів підтримки користувача (користувацькі словники, підказки, повідомлення, допомога тощо) та їх вбудовування в програмний код.

4. Випробування ІК - Usability тестування тестової версії ІК по набору певних показників. Підготовка документації користувача та розробка програми навчання.

Для виконання аналізу діяльності користувача необхідно ретельно продумати і усвідомити сценарій взаємодії програми з користувачем, привівши його до оптимальної (щодо розглянутих показників) системи виконання завдань, і реалізувати ІК у відповідності з цією системою.

Для того, щоб розібратися в технології вирішення завдань користувачем, розробнику необхідно з'ясувати наступні моменти (досліджуючи діяльність користувача):

- яка інформація необхідна користувачеві для вирішення завдання?

- яку інформацію користувач може ігнорувати (не враховувати)?

- спільно з користувачем розділити всю інформацію на сигнальну, що відображається та редагується, пошукову і результуючу;

- які рішення користувачу необхідно приймати в процесі роботи з програмою?

- чи може користувач здійснювати кілька різних дій (вирішувати кілька завдань) одночасно?

- які типові операції виконує користувач при вирішенні задачі?

- що станеться, якщо користувач буде діяти не згідно з написаним алгоритмом, пропускаючи ті чи інші кроки або обходячи їх?

Для оцінки продуктивності використовуються відповідні показники, що перевіряються фахівцями з ергономіки в процесі usability тестування робочого прототипу.

Формування таких показників відбувається в процесі визначення вимог до ІК при вивченні наступних питань:

- що від користувача потрібно в першу чергу?

- скільки інформації, що вимагає обробки, надходить користувачу за період часу?

- які вимоги до точності та швидкості введення інформації?

- на які операції користувач витрачає найбільше часу?

- чим ми можемо полегшити роботу користувача при вирішенні типових завдань?

1.3 Обґрунтування вибору мови програмування

Для написання програми необхідно використати об’єктно-орієнтовану мову програмування. До таких, зокрема, належать Python, C++, C# та Java. Розглянемо детальніше особливості кожної із них.

Python – об'єктно-орієнтована мова програмування високого рівня зі строгою динамічною типізацією. Розроблена в 1990 році Гвідо ван Россумом. Структури даних високого рівня разом із динамічною семантикою та динамічним зв'язуванням роблять її привабливою для швидкої розробки програм, а також як засіб поєднування наявних компонентів. Python підтримує модулі та пакети модулів, що сприяє модульності та повторному використанню коду. Інтерпретатор Python та стандартні бібліотеки доступні як у скомпільованій, так і у вихідній формі на всіх основних платформах. В мові програмування Python підтримується кілька парадигм програмування, зокрема: об'єктно-орієнтована, процедурна, функціональна та аспектно-орієнтована.

Серед основних її переваг можна назвати такі:

- чистий синтаксис (для виділення блоків слід використовувати відступи);

- переносність програм (що властиве більшості інтерпретованих мов);

- стандартний дистрибутив має велику кількість корисних модулів (включно з модулем для розробки графічного інтерфейсу);

- можливість використання Python в діалоговому режимі (дуже корисне для експериментування та розв'язання простих задач);

- стандартний дистрибутив має просте, але разом із тим досить потужне середовище розробки, яке зветься IDLE і яке написане на мові Python;

- зручний для розв'язання математичних проблем (має засоби роботи з комплексними числами, може оперувати з цілими числами довільної величини, у діалоговому режимі може використовуватися як потужний калькулятор).

Python має ефективні структури даних високого рівня та простий, але ефективний підхід до об'єктно-орієнтованого програмування. Елегантний синтаксис Python, динамічна обробка типів, а також те, що це інтерпретована мова, роблять її ідеальною для написання скриптів та швидкої розробки прикладних програм у багатьох галузях на більшості платформ.

Інтерпретатор мови Python і багата Стандартна бібліотека (як вихідні тексти, так і бінарні дистрибутиви для всіх основних операційних систем) можуть бути отримані з сайту Python www.python.org, і можуть вільно розповсюджуватися. Цей самий сайт має дистрибутиви та посилання на численні модулі, програми, утиліти та додаткову документацію.

Інтерпретатор мови Python може бути розширений функціями та типами даних, розробленими на C чи C++ (або на іншій мові, яку можна викликати із C). Python також зручна як мова розширення для прикладних програм, що потребують подальшого налагодження.

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

«Oracle» надає компілятор Java та віртуальну машину Java, які задовольняють специфікації Java Community Process, під ліцензією GNU General Public License.

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

Java вплинула на розвиток J++, що розроблялась компанією «Microsoft». Роботу над J++ було зупинено через судовий позов «Sun Microsystems», оскільки ця мова програмування була модифікацією Java. Пізніше в новій платформі «Microsoft» .NET випустили J#, щоб полегшити міграцію програмістів J++ або Java на нову платформу. З часом нова мова програмування C# стала основною мовою платформи, перейнявши багато чого з Java. J# востаннє включався в версію Microsoft Visual Studio 2005. Мова сценаріїв JavaScript має схожу із Java назву і синтаксис, але не пов'язана із Java.

C++ – мова програмування високого рівня з підтримкою декількох парадигм програмування: об'єктно-орієнтованої, узагальненої та процедурної. Розроблена Б'ярном Страуструпом у 1979 році та названа «Сі з класами». Страуструп перейменував мову у C++ у 1983 р. Стандартна бібліотека C++ включає стандартну бібліотеку Сі з невеликими змінами, які роблять її відповіднішою для мови C++. Інша велика частина бібліотеки C++ заснована на Стандартній Бібліотеці Шаблонів (STL). Вона надає такі важливі інструменти, як контейнери (наприклад, вектори і списки) і ітератори (узагальнені вказівники), що надають доступ до цих контейнерів як до масивів. Крім того, STL дозволяє схожим чином працювати і з іншими типами контейнерів, наприклад, асоціативними списками, стеками, чергами. Використовуючи шаблони, можна писати узагальнені алгоритми, здатні

працювати з будь-якими контейнерами або послідовностями, доступ до членів яких забезпечують ітератори.

При створенні C++ прагнули зберегти сумісність з мовою C. Більшість програм на C справно працюватимуть і з компілятором C++. C++ має синтаксис, заснований на синтаксисі C.

Нововведеннями C++ порівняно з C є:

- підтримка об'єктно-орієнтованого програмування через класи;
- підтримка узагальненого програмування через шаблони;
- доповнення до стандартної бібліотеки;
- додаткові типи даних;
- обробка винятків;
- простори імен;
- вбудовані функції;
- перевантаження операторів;
- перевантаження імен функцій;
- посилання і оператори управління вільно розподіленою пам'яттю.

У 1998 році ратифіковано міжнародний стандарт мови C++: ISO/IEC 14882 «Standard for the C++ Programming Language». Поточна версія цього стандарту — ISO/IEC 14882:2011.

C# – об'єктно-орієнтована мова програмування з безпечною системою типізації для платформи .NET. Розроблена Андерсом Гейлсбергом, Скотом Вілтамутом та Пітером Гольде під егідою Microsoft. Синтаксис C# близький до C++ і Java. Мова має строгу статичну типізацію, підтримує поліморфізм, перевантаження операторів, вказівники на функції-члени класів, атрибути, події, властивості, винятки, коментарі у форматі XML. Перейнявши багато що від своїх попередників – мов C++, Delphi, Модула і Smalltalk – C#, спираючись на практику їхнього використання, виключає деякі моделі, що зарекомендували себе як проблематичні при розробці програмних систем: так, C# не підтримує множинне спадкування класів (на відміну від C++) або виведення типів (на відміну Haskell).

C# розроблялась як мова програмування прикладного рівня для CLR і тому вона залежить, перш за все, від можливостей самої CLR. Це стосується, перш за все, системи типів C#. Присутність або відсутність тих або інших виразних особливостей мови диктується тим, чи може конкретна мовна особливість бути трансльована у відповідні конструкції CLR. Так, з розвитком CLR від версії 1.1 до 2.0 значно збагатився і сам C#; подібної взаємодії слід чекати і надалі. (Проте ця закономірність буде порушена з виходом C# 3.0, що є розширеннями мови, що не спираються на розширення платформи .NET.) CLR надає C#, як і всім іншим .NET-орієнтованим мовам, багато можливостей, яких позбавлені «класичні» мови програмування. Наприклад, збірка сміття не реалізована в самому C#, а проводиться CLR для програм, написаних на C# точно так, як і це робиться для програм на VB.NET, J# тощо.

MicrosoftVisualStudio - лінійка продуктів компанії Майкрософт, що включають інтегроване середовище розробки програмного забезпечення та ряд інших інструментальних засобів. Дані продукти дозволяють розробляти як консольні додатки, так і додатки з графічним інтерфейсом, в тому числі з підтримкою технології Windows Forms, а також веб-сайти, веб-додатки, веб-служби як в рідному, так і в керованому кодах для всіх платформ, підтримуваних Microsoft Windows, Windows Mobile, Windows CE, .NETFramework, .NETCompactFramework і MicrosoftSilverlight.

VisualStudio включає в себе редактор вихідного коду з підтримкою технології IntelliSense і можливістю найпростішого рефакторинга коду. Вбудований відладчик може працювати як відладчик рівня вихідного коду, так і як відладчик машинного рівня. Решта вбудовувані інструменти включають в себе редактор форм для спрощення створення графічного інтерфейсу програми, веб-редактор, дизайнер класів і дизайнер схеми бази даних. Visual Studio дозволяє створювати і підключати сторонні додатки (плагіни) для розширення функціональності практично на кожному рівні, включаючи додавання підтримки систем контролю версій вихідного коду (як наприклад, Subversion і Visual SourceSafe), додавання нових наборів інструментів (наприклад, для редагування та візуального проектування

коду на предметно-орієнтованих мовах програмування або інструментів для інших аспектів циклу розробки програмного забезпечення (наприклад, клієнт Team Explorer для роботи з Team Foundation Server).

Для розробки програмного продукту було обрано мову програмування Microsoft Visual C#, відповідно, платформу .NET, інтегроване середовище розробки Microsoft Visual Studio 2019.

Вибір платформи .NET зумовлений рядом переваг:

- вся платформа .NET ґрунтується на єдиній обєктно-орієнтованій моделі;

- в склад платформи .NET входить «збиральник сміття», який звільняє ресурси, що захищає програми від втрат памяті і від необхідності звільняти ресурси;

- будь-яка програма, розроблена з допомогою .NET є автономною, в тому розумінні, що не залежить від інших програм та від ОС;

- встановлення програми може бути проведене звичайним копіюванням файлів;

- використання безпечних типів даних, що підвищує надійність програми та сумісність;

- програма взаємодіє з єдиною моделлю обробки помилок;

- програми, написані на різних мовах можуть легко взаємодіяти;

- абсолютно всі помилки оброблюються механізмом виключних ситуацій, що дозволяє запобігти неоднозначностям, які виникали інколи при програмуванні під Win32;

- зручний спосіб повторного використання коду.

Проте є і деякі недоліки, які не впливають на досягнення поставлених вимог та остаточного результату:

- швидкість виконання коду менша приблизно на 5%, порівняно з мовою програмування C++;

- необхідна наявність бібліотеки FrameWork.

Мова програмування C# найбільше відповідає поставленим вимогам, у порівнянні з іншими мовами програмування, які входять до складу Visual Studio 2019 саме тому її використано для розробки програмного продукту.

2 Спеціальна частина

2.1 Опис алгоритму створення програмного засобу

Проектування програмного забезпечення необхідно почати з побудови таких діаграм:

- діаграи класів;

- варіантів використання;

- послідовності.

- взаємодії;

Діаграма класів — статичне представлення структури моделі. Відображає статичні (декларативні) елементи, такі як класи, типи, їх зміст та відношення. Діаграма класів, також, може містити позначення для пакетів та може містити позначення для вкладених пакетів. Також, діаграма класів може містити позначення деяких елементів поведінки, однак їх динаміка розкривається в інших типах діаграм.

Діаграма класів (classdiagram) служить для представлення статичної структури моделі системи в термінології класів об'єктно-орієнтованого програмування. На цій діаграмі показують класи, інтерфейси, об'єкти й кооперації, а також їхні відносини.

Діаграма варіантів використання – це граф спеціального вигляду, який є графічною нотацією для представлення конкретних варіантів використання, акторів, можливо деяких інтерфейсів, і відносин між цими елементами. При цьому окремі компоненти діаграми можуть бути поміщені в прямокутник, який позначає проектовану систему в цілому. Слід зазначити, що відносинами даного графа можуть бути тільки деякі фіксовані типи взаємозв'язків між акторами і варіантами використання, які в сукупності описують сервіси або функціональні вимоги до модельованої системи.

Суть діаграми варіантів використання полягає в наступному. Проектована система представляється у вигляді безлічі суті або акторів, що взаємодіють з системою за допомогою варіантів використання. При цьому актором (actor) або дійовою особою називається будь-яка сутність, що взаємодіє з системою ззовні. Це може бути людина, технічний пристрій, програма або будь-яка інша система, яка може служити джерелом дії на модельовану систему так, як визначить сам розробник. Варіант використання служить для опису сервісів, які система надає актору. Діаграма варіантів використання може доповнюватися текстом пояснення, який розкриває сенс або семантику складових її компонентів.

Діаграма послідовності — в UML, діаграма послідовності відображає взаємодії об'єктів впорядкованих за часом. Зокрема, такі діаграми відображають задіяні об'єкти та послідовність відправлених повідомлень.

Діаграми послідовностей - це відмінний засіб документування поведінки системи, деталізації логіки сценаріїв використання, але є ще один спосіб – використовувати діаграми взаємодії. Діаграма взаємодії показує потік повідомлень між об'єктами системи і основні асоціації між ними і по суті, як вже було сказано вище, є альтернативою діаграми послідовностей. Слід зазначити, що використання діаграми послідовностей або діаграми взаємодії - особистий вибір кожного проектувальника і залежить від індивідуального стилю проектування. На позначеннях, що застосовуються на діаграмі взаємодії, думаємо, не варто зупинятися детально. Тут все стандартно: об'єкти позначаються прямокутниками з підкресленими іменами (щоб відрізнити їх від класів), асоціації між об'єктами вказуються у вигляді з'єднувальних ліній, над ними може бути зображена стрілка із зазначенням назви повідомлення та його порядкового номера. Необхідність номеру повідомлення пояснюється дуже просто - на відміну від діаграми послідовностей, час на діаграмі взаємодії не показується у вигляді окремого вимірювання.

Діаграма взаємодії. При об’єднанні схем прецедентів можна суттєво покращити документацію і зробити взаємодію більш зрозумілою. Формується

діаграма взаємодії, яка дозволяє виявити такі подробиці сценарію, які не завжди можна побачити при читанні тексту.

Оскільки для кожної задачі створюється ряд прецедентів різної складності, UML дозволяє об’єднати їх в пакети.

Пакет подібний каталогу або папці, це є набір за певною характеристикою об’єктів, що моделюються. Один прецедент може бути присутнім у різних пакетах.

Окрім прецедентів, документ вимог включає передбачення замовника проекту, умови, обмеження, вимоги до апаратних засобів та операційної системи. Вимоги до проекту – це саме те, що хоче конкретний замовник. Якщо проект передбачає взаємодію з існуючими системами, то необхідно розібратись з цими системами, як вони працюють. Необхідно вирішити яка система буде сервером, а яка клієнтом. Чи прийдеться налагоджувати нову систему під існуючий стандарт? Як довго буде використовуватись нова система? Обов’язково необхідно зафіксувати умови та обмеження, які можуть виникнути при взаємодії з іншими системами.

Переважно замовник вже має певну операційну систему та апаратну платформу і хоче щоб нова система працювала саме під його ОП. Це необхідно врахувати у вимогах.

Складаючи бюджет та графік дотримуйтесь наступних принципів:

- враховуйте можливість перероcходу фондів;

- враховуйте можливість перероcходу часу.

Тому роботу необхідно розподілити за пріоритетом. Все і вчасно закінчити не вдасться, треба бути до цього готовим. Головне, щоб вчасно був готовий працездатний екземпляр програми на рівні першої версії. Точний час реалізації стане відомий тільки на етапі тестування. Крім того бажано зарезервувати 20-25% часу на всяк випадок, якщо по ходу проекту виникнуть непередбачені ситуації.

Даний програмний засіб призначений для аналізу зображення водойми на предмет забруднення її сміттям. Діаграма класів даного програмного засобу має вигляд продемонстрований на рисунку 2.1.

Рисунок 2.1 – Діаграма класів

Враховуючи те, програма не має розподілу за користувачами з різними правами доступу, тобто передбачено лише одного користувача, діаграма варіантів використання має вигляд продемонстрований на рисунку 2.2.

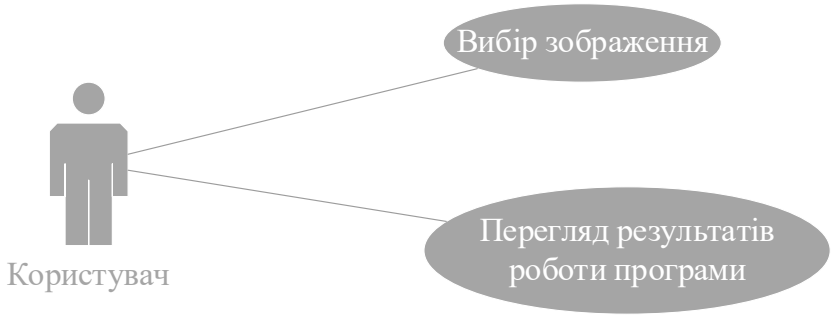

Рисунок 2.2 – Діаграма варіантів використання

Згідно рис. 2.2, користувачу надано можливість обрати необхідне зображення та отримати результат.

Для демонстрації взаємодії користувача з проектованим програмним забезпеченням побудовано діаграму взаємодії зображену на рисунку 2.3.

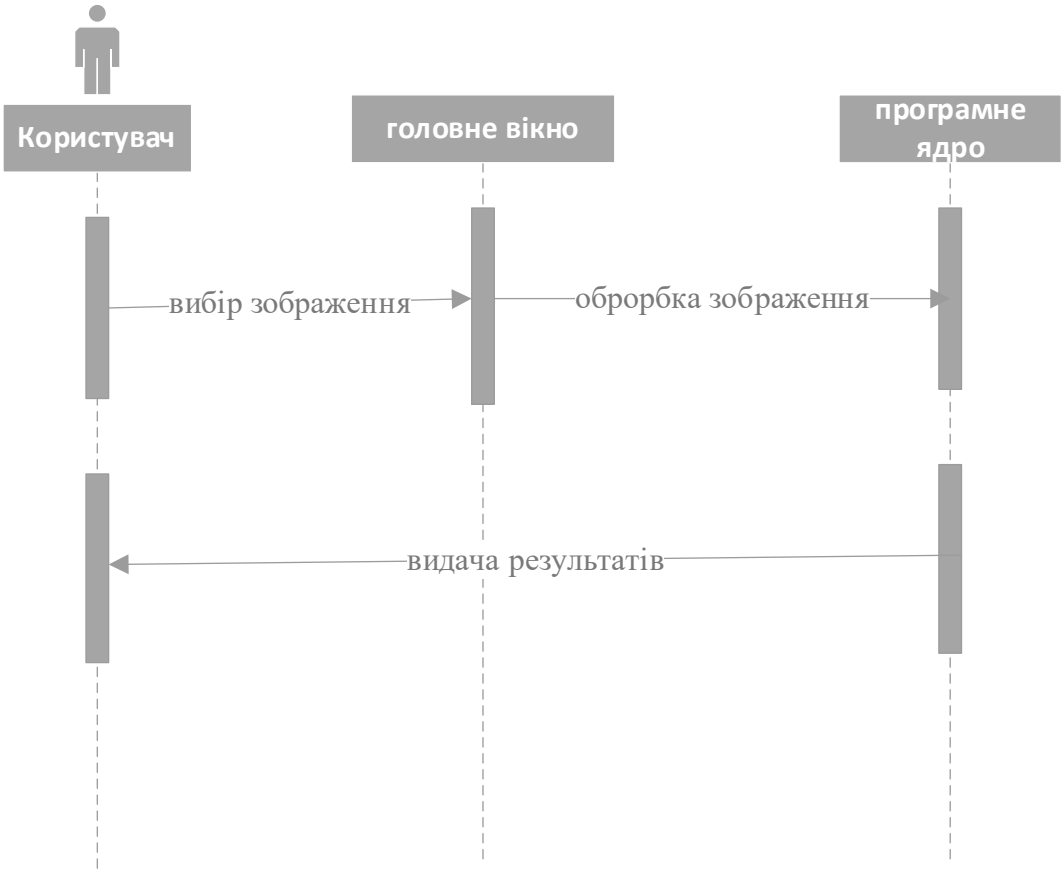

Рисунок 2.3 – Діаграма взаємодії

Діаграма взаємодії демонструє взаємодію між користівачем та програмним забезпеченням, а саме як користувач впливає на роботу програми та як компоненти програми взаємодіють між собою у відповідь на дію користувача.

Також на рисунку 2.3 вказано видимі користувачеві компоненти програмного забезпечення з якими користувач вступає в заємодію.

Наступним кроком необхідно зрозуміти послідовність дій програми які необхідні для відображення результату виконання програми. Для цого будується діаграма послідовностей зображена на рисунку 2.4.

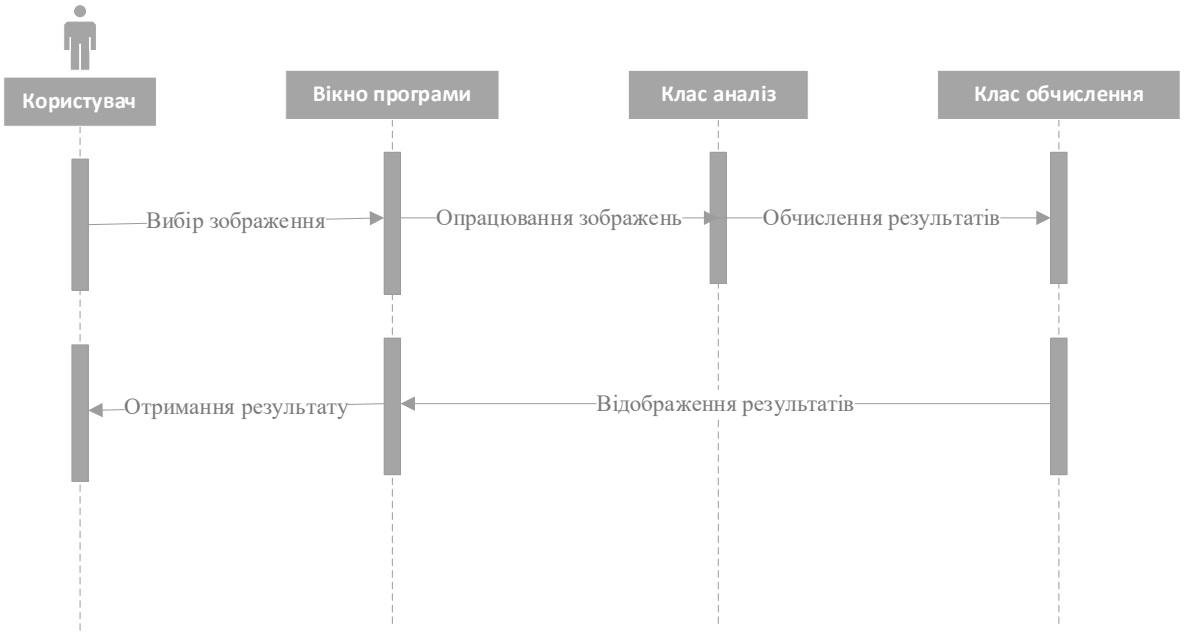

Рисунок 2.4 – Діаграма послідовності

Діаграма послідовностей демонструє послідовність операцій що виконуються програмою у відповідь на дію користувача. Також на рисунку 2.4 продемонстровано всі вузлові компоненти програмного забезпечення між якими відбувається взаємодія.

Проаналізувавши створені діаграми послідовності та взаємодії можна побудувати діаграму станів зображену на рисунку 2.5.

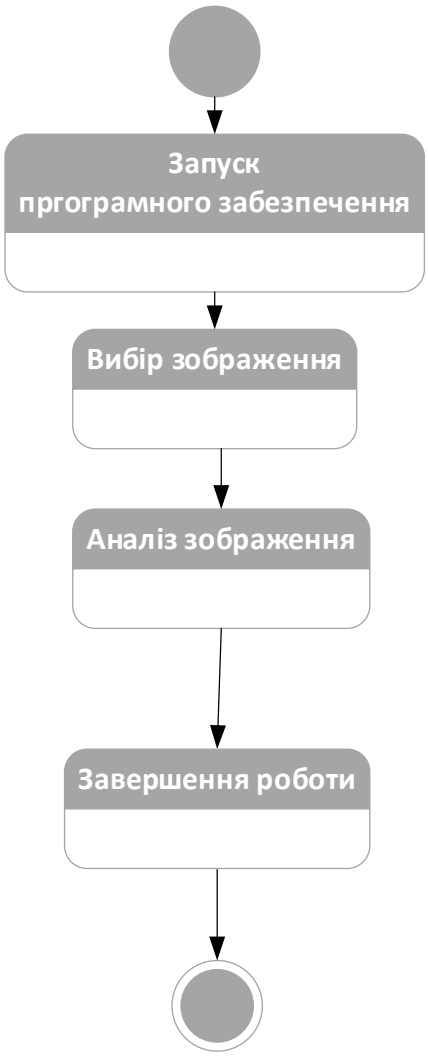

Рисунок 2.5 – Діаграма станів

Діаграма станів демонструє стани програмного забезпечення в якому може перебувати проектоване програмне забезпечення та переходи між ними. Як видно з рисунка 2.5 програмний модуль має два функціональні стани та два стани початку та завершення роботи.

2.2 Опис засобів реалізації

Для реалізації даного програмного модулю було використано такі елементи, як класи, об’єкти класу, методи, події та елементи необхідні для відображення тексту та проектування інтерфейсу.

Клас - це спеціальна конструкція, яка використовується для групування пов'язаних змінних та функцій. При цьому, згідно з термінологією ООП, глобальні змінні класу (члени-змінні) називаються полями даних, а члени-функції називають методами класу. Створений та ініціалізований екземпляр класу називають об'єктом класу. На основі одного класу можна створити безліч об'єктів, що відрізнятимуться один від одного своїм станом (значеннями полів).

Клас можна порівнювати з формою для випічки печива — форма одна, а печива можна випекти безліч. Печиво — це конкретні об'єкти, екземпляри класу печиво, яке може бути з різною начинкою. Поля дозволяють вмістити дані про певний реальний об'єкт, а методи здійснювати обробку цих даних. Наприклад, можна створити загальний клас Людина з полями Ім'я та Прізвище, рік народження, професія, зарплата. При створенні ж на основі класу конкретного екземпляру, дані поля заповнюються конкретними даними про певну людину. Обробкою цих даних можуть займатися відповідні методи. Наприклад, можна створити метод для обчислення віку людини, тощо.

На основі класів можна створювати підкласи, які успадковують властивості та поведінку батьківських класів. Таким чином можна створити цілу ієрархію класів. Різні мови дещо по різному реалізовують механізм успадкування. Існує множинне та одинарне успадкування. Множинне — це, коли підклас створюється на основі кількох безпосередніх батьків (як то в мові програмування C++). Одинарне успадкування — це коли клас може мати одного безпосереднього батька (мова програмування Java). Надкласи можуть мати свої надкласи, підкласи можуть також бути надкласами для певних класів.

Метод - підпрограма (процедура, функція), що використовується виключно разом із класом (методи класу) або з об'єктом (методи екземпляра).

Розрізняють прості методи і статичні методи (методи класу):

- прості методи мають доступ до даних об'єкта (конкретного екземпляра даного класу);

- статичні методи не мають доступу до даних об'єкта і для їх використання не потрібно створювати екземпляри (даного класу).

Методи і властивості, які не прив'язані до конкретного екземпляру об'єкта, називають «статичними». Їх записують прямо в саму функцію-конструктор.

Подібно до процедури в процедурних мовах програмування, метод, як правило, складається із послідовності операцій для виконання дії, множини вхідних параметрів для налаштування цієї дії, та, можливо, вихідного значення деякого типу (значення, що повертається).

Призначення методів полягає в здійсненні певних дій над полями класу (змінними-членами) та надання механізму доступу до цих полів даних, що інкапсулюються в об'єкті або класі.

Залежно від способу використання, методи поділяються на:

- змістовні — надаються клієнтам класу і визначають його інтерфейс, звичайно є публічними;

- спеціальні — викликаються автоматично при створенні (конструктори), знищенні (деструктори), копіюванні (конструктори копіювання), перетворенні типу тощо;

- допоміжні — викликаються з змістовних та спеціальних методів, звичайно не надаються клієнтам класу і є захищеними або приватними.

Залежно від впливу на стан об'єкта, методи поділяються на:

- конструктори — встановлюють початковий стан об'єкта;

- деструктори — скидають стан об'єкта;

- селектори (геттери) — надають значення атрибута;

- модифікатори (сеттери) — встановлюють значення атрибута;

- ітератори — надають послідовний доступ до множини атрибутів.

Події - дія яка розпізнається програмним забезпеченням та оброблюється за допомогою певних інструкцій. Комп'ютерні події можуть бути згенеровані або ініціалізовані системно, користувачем або іншими способами. Як правило, події обробляються синхронно з програмою, тобто, програмне забезпечення може мати один або кілька виділених місць, де обробляються події, часто називається цикл подій. Події виникають при виконанні користувачем певних дій, наприклад, натисканням клавіш на клавіатурі. Деякі події виникають по таймеру, наприклад

перезапуск системи. Програмне забезпечення може також викликати свій власний набір подій в цикл обробки подій, наприклад, повідомити про завершення завдання. Деякі програми кардинально змінюють свою поведінку у відповідь на події, вони називаються подійно-орієнтованими.

Обробник події - це підпрограма, яка опрацьовує матеріали, отримані з програми. Обробник подій певним чином реагує на події, та починає виконувати дії які згенерувала та чи інша подія. Обробники можуть по-різному опрацьовувати події, все залежить від реалізації.

Кожна подія є фрагмент інформації на рівні додатків від базової структури. В GUI події включають в себе натискання клавіші, вибір дій та таймер. На низькому рівні, події можуть представляти зчитування потоку з файла або мережі. Обробники подій є центральним поняттям програмуванні будь-якої програми. Всі події зберігаються в диспетчері подій.

2.3 Порівняльний аналіз реалізованого програмного засобу та програм-аналогів

Для даного програмного забезпечення анлізів в настількі вузькій спеціальності на ринку апаратнопрограмних та програмних комплексах на момент розробки програмног забезпеченн для кваліфікаційної роботи невиявлено, найблищим за принципом дії є програми для визначення якості оброблення земельних ділянок та сільскогосподарських культур які вирощують фермери.

На ринку існують програмні комплекси для моделювання забруднень річок, зон затоплень, наслідків гідротехнічних аварій, прориву гребель, прогнозу паводків і повеней в складній системі річок та каналів.

Перевагами таких програмних комлексів є моделювання забруднень річок що дає можливість підготовити комплекс рішень необхідний для ліквідації. Мета програм аналогів проаналізувати всі можливі варіанти розвитку подій пов’язаних з

водоймищами та грунтовими водами, прогнозувати поведення гідросфери земної кулі.

Розроблюване програмне забезпечення в даній кваліфікаційній роботі має на меті спостереження за водоймами в режимі реального часу для вчасного реагування на зміни які не можливо завбачити або мають низьку ймовірність статися, а саме: велике скупчення сміття яке було викинуто у водоймище, цвітіння води, зливання відходів до водойми.

2.4 Інструкція роботи користувача

Для початку роботи з розобленою програмою необхідно впевнитись що персональний комп’ютер відповідає мінімальним системним вимогам. Наступним кроком запускаємо виконавчий файл Water.exe. Далі користувача зустрічає головне вікно програми яке зображено на рисунку 2.6.

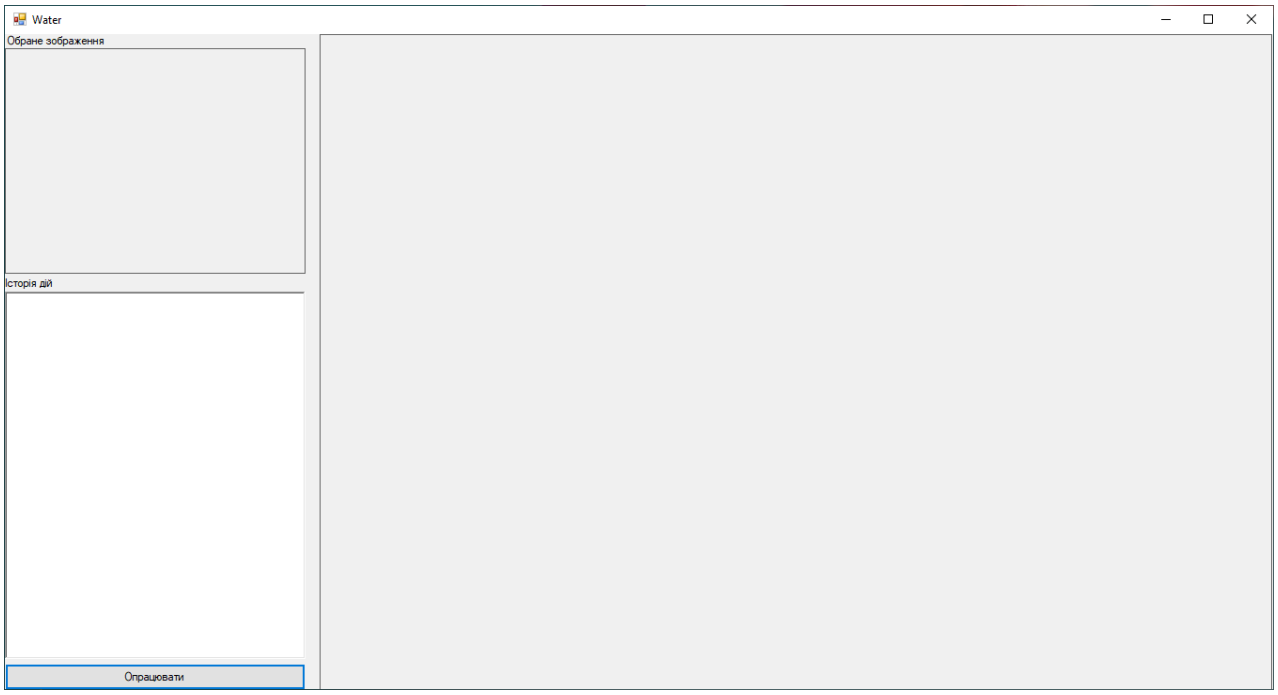

Рисунок 2.6 – головне вікно програми

Головне вікно містить одну функціональну кнопку та одну інтерактивну зону. Натиснувши на інтерактивну зону показану на рисунку 2.7 користувачу відкривається вікно для вибору зображення для аналізу програмою продемонстроване на рисунку 2.8.

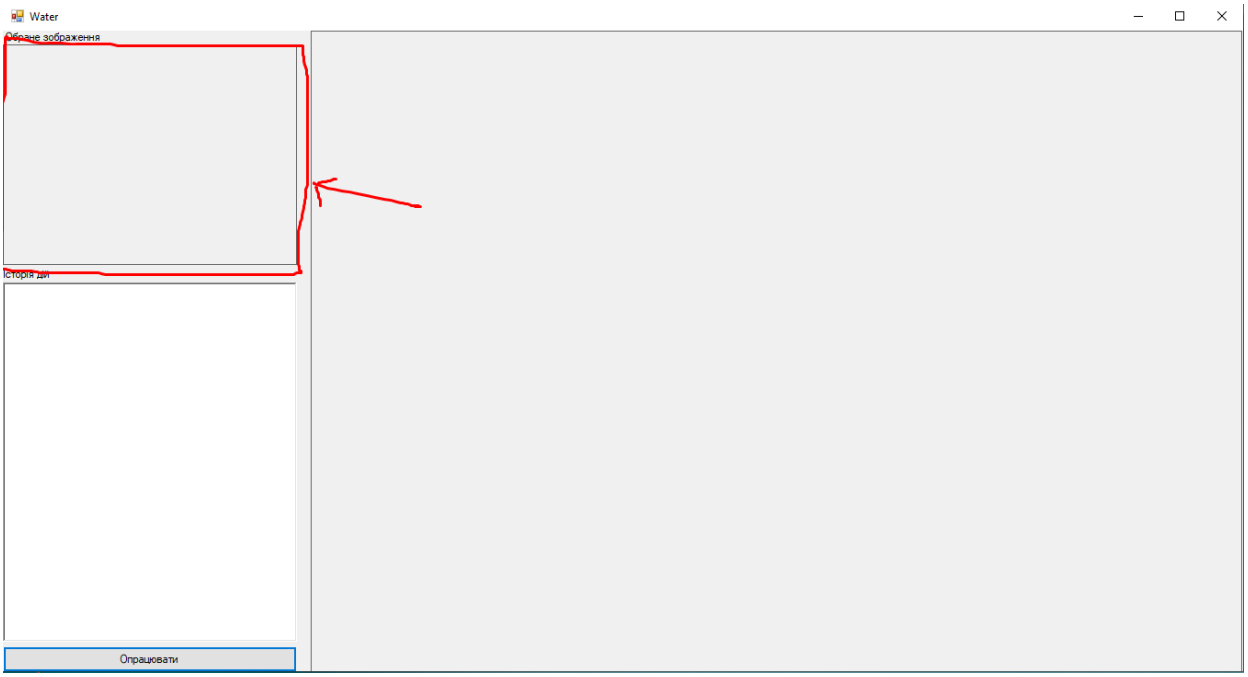

Рисунок 2.7 – інтерактивна зона

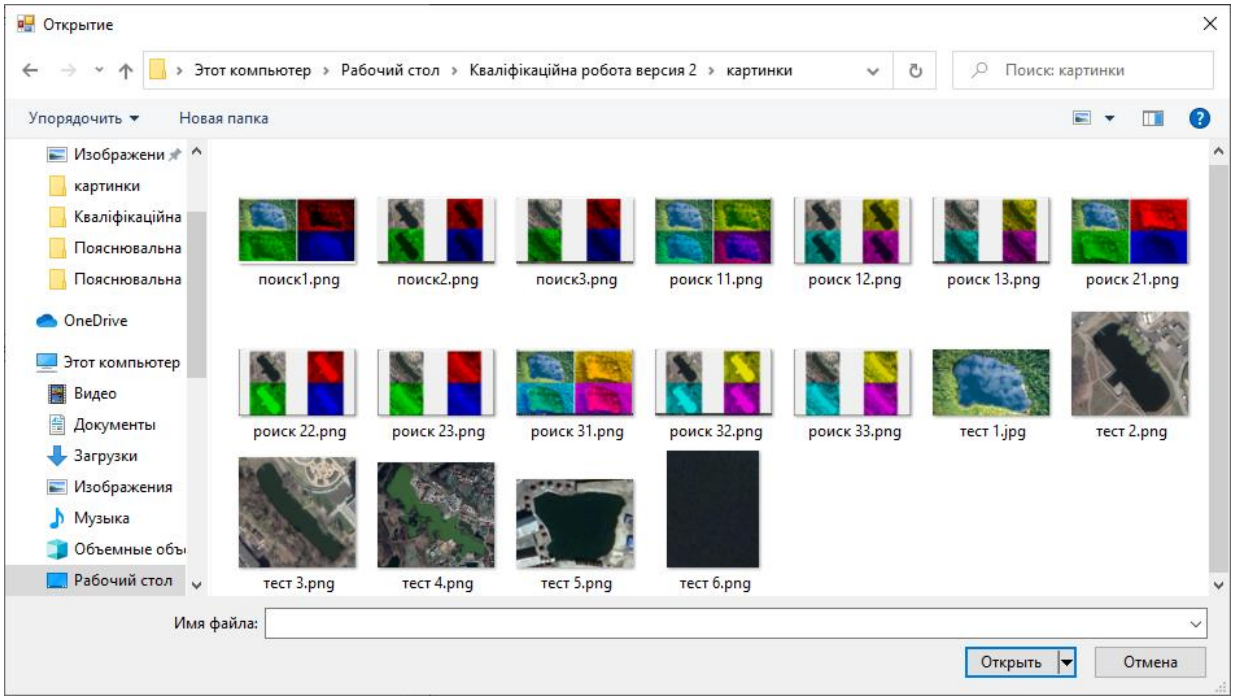

Рисунок 2.8 – вікно вибору зображення

Після того як користувач обере необхідне зображення воно відобразиться в інтерактивній зоні як показано на рисунку 2.9.

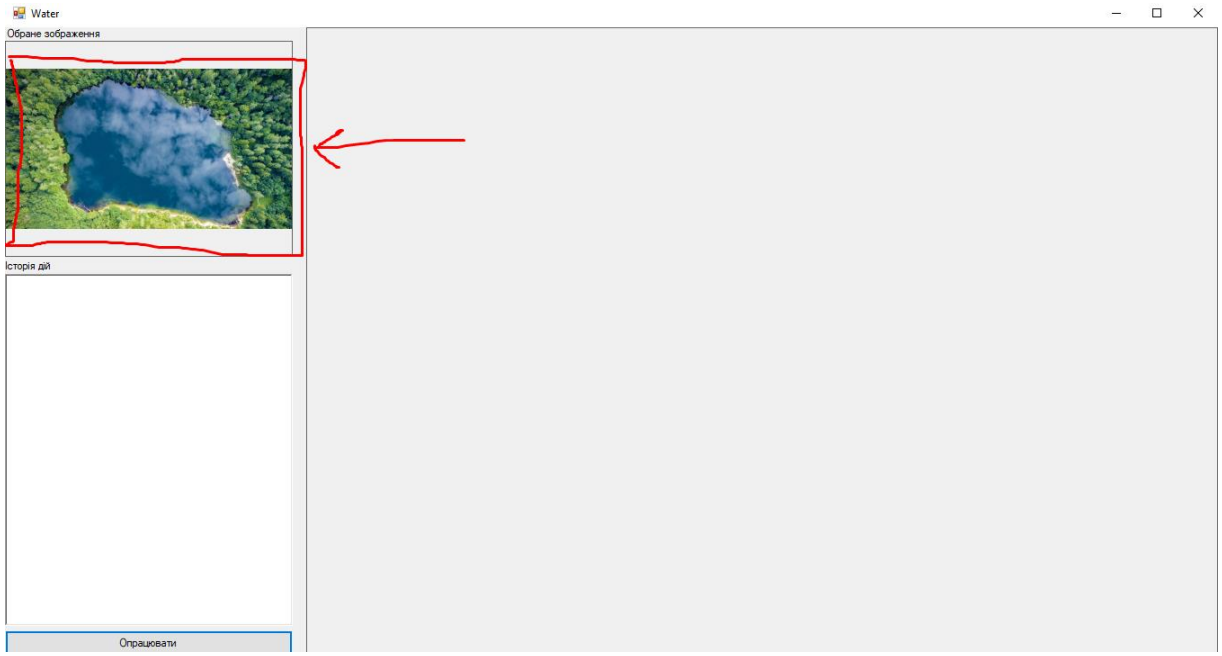

Рисунок 2.9 – інтерактивна зона з обраним зображенням

Наступним кроком користувачу необхідно натиснути кнопку «Опрацювати», яку продемонстровано на рисунку 2.10.

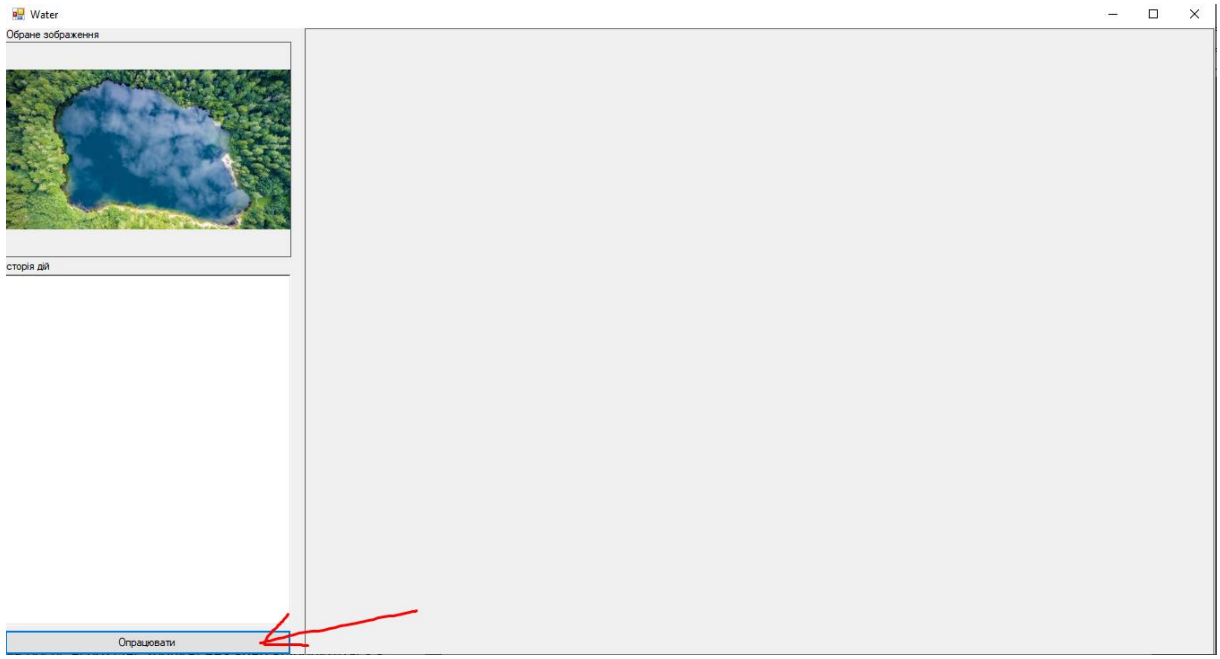

Рисунок 2.10 – кнопка «Опрацювати»

Результат роботи програми, а саме обчислення продемонстровано на рисунку 2.11, та графічне відображення зон забруднення водойми продемонстровано на рисунку 2.12.

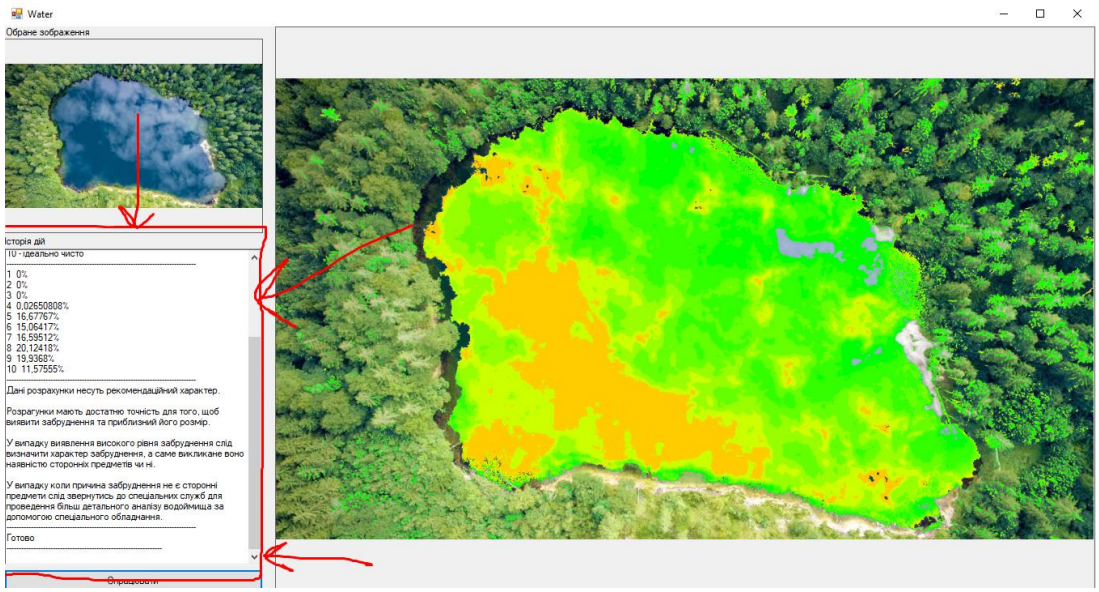

Рисунок 2.11 – зона результатів обчислення

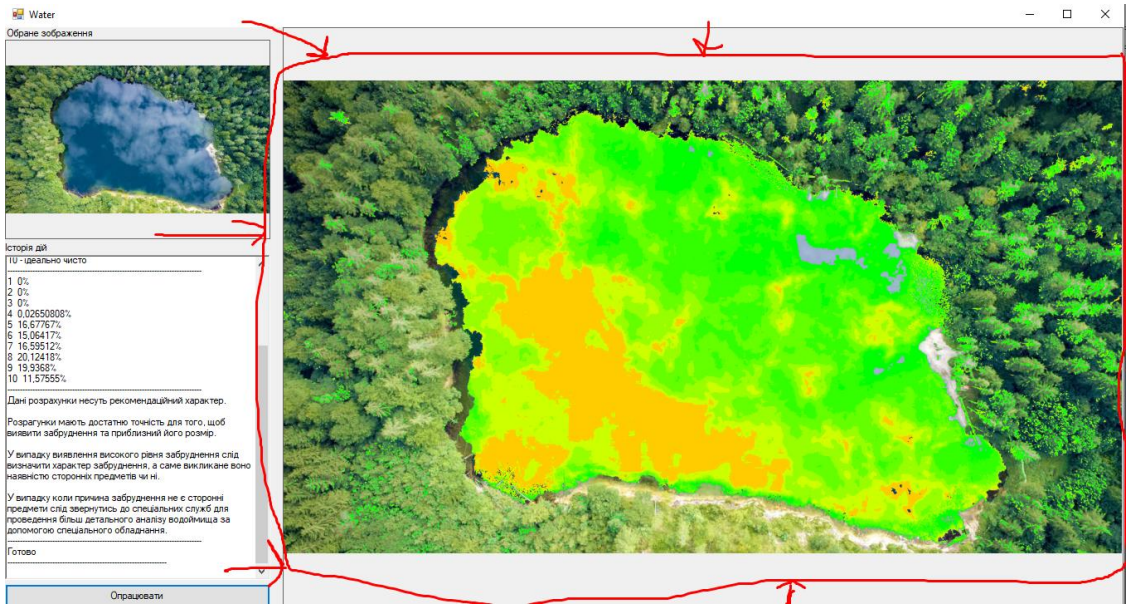

Рисунок 2.12 – зона графічно відображення результатів аналізу

Для повторного використання програми необхідно повторити дії починаючи з натискання на інтерактивну зону.

Для завершення роботи необхідно наитснути на кнопку «закрити», або скористатись іншими відомими користувачу способами.

2.5 Тестування реалізованого програмного засобу

Тестування програмного забезпечення — процес перевірки відповідності заявлених до продукту вимог і реально реалізованої функціональності, здійснюваний шляхом спостереження за його роботою в штучно створених ситуаціях і на обмеженому наборі тестів, обраних певним чином.

Може оцінюватись:

- відповідність вимогам, якими керувалися проектувальники та розробники
- правильна відповідь для усіх можливих вхідних даних
- виконання функцій за прийнятний час
- практичність
- сумісність із програмним забезпеченням та операційними системами
- відповідність задачам замовника.

Оскільки число можливих тестів навіть для нескладних програмних компонентів практично нескінченне, тому стратегія тестування полягає в тому, щоб провести всі можливі тести з урахуванням наявного часу та ресурсів. Як результат програмне забезпечення (ПЗ) тестується стандартним виконанням програми з метою виявлення багів (помилок або інших дефектів).

Тестування ПЗ може надавати об'єктивну, незалежну інформацію про якість ПЗ, ризики відмови, як для користувачів, так і для замовників.

Тестування може проводитись, як тільки створено виконуваний код (навіть частково завершений). Процес розробки зазвичай передбачає, коли та як буде відбуватися тестування. Наприклад, при поетапному процесі, більшість тестів відбувається після визначення системних вимог і тоді вони реалізуються в тестових програмах. На противагу цьому, відповідно до вимог гнучкої розробки ПЗ, програмування і тестування часто відбувається одночасно.

Тестування — це одна з технік контролю якості, що включає в себе

- Планування робіт (Test Management)
- Проектування тестів (Test Design)
- Виконання тестування (Test Execution)

- Аналіз отриманих результатів (Test Analysis).

Верифікація (Verification) — це процес оцінки системи або її компонентів із метою визначити чи задовольняють результати поточного етапу розробки умовам, сформованим на початку цього етапу. Тобто чи виконуються цілі, терміни, завдання з розробки проекту, визначені на початку поточної фази. Валідація (Validation) — це визначення відповідності розроблюваного програмного забезпечення очікуванням і потребам користувача, вимогам до системи.

План Тестування (Test Plan) — це документ, що описує весь обсяг робіт із тестування, починаючи з опису об'єкта, стратегії, розкладу, критеріїв початку і закінчення тестування, до необхідного в процесі роботи обладнання, спеціальних знань, а також оцінки ризиків із варіантами їх вирішення.

Тест дизайн (Test Design) — це етап процесу тестування програмного забезпечення, на якому проектуються і створюються тестові випадки (тест кейси), відповідно до визначених раніше критеріями якості та цілями тестування.

Тестовий випадок (Тест кейс/Test Case) — це документ, що описує сукупність кроків, конкретних умов і параметрів, необхідних для перевірки реалізації тестованої функції або її частини.

Баг/Дефект Репорт (Bug Report) — це документ, що описує ситуацію або послідовність дій (Steps), що призвела до некоректної роботи об'єкта тестування (Misbehavior), із зазначенням причин та очікуваного результату (Expected Result).

Тестове Покриття (Test Coverage) — це одна з метрик оцінки якості тестування, що представляє із себе щільність покриття тестами вимог або коду, що виконується.

Деталізація Тест Кейсів (Test Case Specification) — це рівень деталізації опису тестових кроків і необхідного результату, при якому забезпечується розумне співвідношення часу проходження до тестового покриття.

Час Проходження Тест Кейса (Test Case Pass Time) — це час від початку проходження кроків тест кейса до отримання результату тесту.

Статичне та динамічне тестування

Тестова діяльність, що пов'язана з аналізом результатів розробки програмного забезпечення, називається статичним тестуванням. Воно передбачає перевірку програмних кодів, контроль та перевірку програми без запуску на комп'ютері. Тестова діяльність, що передбачає експлуатацію програмного продукту, називається динамічним тестуванням. Динамічне та статичне тестування доповнюють одне одного.

На етапі статичного тестування перевіряється вся документація, отримана як результат життєвого циклу програми. Це і технічне завдання, і специфікація, і вихідний текст програми на мові програмування. Вся документація аналізується на предмет дотримання стандартів програмування. У результаті статичної перевірки встановлюється, наскільки програма відповідає заданим критеріям та вимогам замовника. Усунення неточностей та помилок у документації — запорука того, що створюваний програмний засіб має високу якість.

Динамічні методи застосовуються в процесі безпосереднього виконання програми. Коректність програмного засобу перевіряється на безлічі тестів або наборів підготовлених вхідних даних. При прогоні кожного тесту збираються та аналізуються дані про відмови та збої в роботі програми.

Тестування «білої скриньки»

Відома: внутрішня структура програми.

Досліджуються: внутрішні елементи програми і зв'язки між ними.

Об'єктом тестування тут є не зовнішня, а внутрішня поведінка програми.

Перевіряється коректність побудови всіх елементів програми та правильність їхньої взаємодії один з одним. Зазвичай аналізуються керуючі зв'язки елементів, рідше — інформаційні зв'язки. Тестування за принципом «білої скриньки» характеризується ступенем, в якому тести виконують або покривають логіку (вихідний текст) програми.

Особливості тестування «білої скриньки»

Зазвичай тестування «білої скриньки» засноване на аналізі керуючої структури програми. Програма вважається повністю перевіреною, якщо проведено вичерпне тестування маршрутів (шляхів) її графа управління.

У цьому випадку формуються тестові варіанти, в яких:

- Гарантується перевірка всіх незалежних маршрутів програми.
- Знаходяться гілки True, False для всіх логічних рішень.
- Виконуються всі цикли (у межах їхніх кордонів та діапазонів).
- Аналізується правильність внутрішніх структур даних.

Недоліки тестування «білої скриньки»:

- Кількість незалежних маршрутів може бути дуже велика.
- Повне тестування маршрутів не гарантує відповідності програми вихідним вимогам до неї.
- У програмі можуть бути пропущені деякі маршрути.
- Не можна виявити помилки, поява яких залежить від даних.

Переваги тестування «білої скриньки» пов'язані з тим, що принцип «білої скриньки» дозволяє врахувати особливості програмних помилок:

- Кількість помилок мінімально в «центрі» і максимально на «периферії» програми.
- Попередні припущення про ймовірність потоку керування або даних у програмі часто бувають некоректними. У результаті типовим може стати маршрут, модель обчислень за яким опрацьована слабо.
- При записі алгоритму програмного забезпечення у вигляді тексту на мові програмування можливе внесення типових помилок трансляції (синтаксичних та семантичних).
- Деякі результати в програмі залежать не від вихідних даних, а від внутрішніх станів програми.

Кожна з цих причин є аргументом для проведення тестування за принципом «білої скриньки». Тести «чорної скриньки» не зможуть реагувати на помилки таких типів.

Тестування «чорної скриньки»

Відомі: функції програми.

Досліджується: робота кожної функції на всій області визначення.

Основне місце програми тестів «чорної скриньки» — інтерфейс ПЗ.

Ці тести демонструють:

- Як виконуються функції програми.
- Як приймаються вихідні дані.
- Як виробляються результати.
- Як зберігається цілісність зовнішньої інформації.

При тестуванні «чорної скриньки» розглядаються системні характеристики програм, ігнорується їхня внутрішня логічна структура. Вичерпне тестування, як правило, неможливе. Наприклад, якщо в програмі 10 вхідних величин і кожна приймає по 10 значень, то кількість тестових варіантів становитиме 10^{10}. Тестування «чорної скриньки» не реагує на багато особливостей програмних помилок.

Тестування «чорної скриньки» (функціональне тестування) дозволяє отримати комбінації вхідних даних, які забезпечують повну перевірку всіх функціональних вимог до програми. Програмний виріб тут розглядається як «чорна скринька», чию поведінку можна визначити тільки дослідженням його входів та відповідних виходів. При такому підході бажано мати:

- Набір, утворений такими вхідними даними, які призводять до аномалій у поведінці програми (назвемо його ІТ);
- Набір, утворений такими вхідними даними, які демонструють дефекти програми (назвемо його ОТ).

Будь-який спосіб тестування «чорної скриньки» повинен:

- Виявити такі вхідні дані, які з високою ймовірністю належать набору ІТ;
- Сформулювати такі очікувані результати, які з високою імовірністю є елементами набору ОТ.

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

Тестування «чорної скриньки» забезпечує пошук наступних категорій помилок:

- Некоректних чи відсутніх функцій;

- Помилок інтерфейсу;
- Помилок у зовнішніх структурах даних або в доступі до зовнішньої бази даних;
- Помилок характеристик (необхідна ємність пам'яті і т. д.);
- Помилок ініціалізації та завершення.

Подібні категорії помилок способами «білої скриньки» не виявляються.

3 Охорона праці

3.1 Характеристика умов праці програміста

Науково-технічний прогрес вніс серйозні зміни в умови виробничої діяльності робітників розумової праці. Їх праця стала більш інтенсивним, напруженим, які вимагають значних витрат розумової, емоційної і фізичної енергії. Це зажадало комплексного рішення проблем ергономіки, гігієни і організації праці, регламентації режимів праці та відпочинку.

В даний час комп'ютерна техніка широко застосовується у всіх областях діяльності людини. При роботі з комп'ютером людина піддається дії ряду небезпечних і шкідливих виробничих факторів: електромагнітних полів (діапазон радіочастот: ВЧ, УВЧ і СВЧ), інфрачервоного і іонізуючого випромінювань, шуму і вібрації, статичної електрики і ін.

Робота з комп'ютером характеризується значною розумовою напругою і нервово-емоційним навантаженням операторів, високою напруженістю зорової роботи і достатньо великим навантаженням на м'язи рук при роботі з клавіатурою ЕОМ. Велике значення має раціональна конструкція і розташування елементів робочого місця, що важливо для підтримки оптимальної робочої пози людини-оператора.

У процесі роботи з комп'ютером необхідно дотримувати правильний режим праці та відпочинку. В іншому випадку у персоналу наголошуються значна напруга зорового апарату з появою скарг на незадоволеність роботою, головні болі, дратівливість, порушення сну, втому і хворобливі відчуття в очах, в поясниці, в області шиї і руках.

3.2 Вимоги до виробничих приміщень

3.2.1 Забарвлення і коефіцієнт віддзеркалення

Забарвлення приміщень і меблів повинні сприяти створенню сприятливих умов для зорового сприйняття, гарного настрою.

Джерела світла, такі як світильники і вікна, які дають віддзеркалення від поверхні екрану, значно погіршують точність знаків і тягнуть за собою перешкоди фізіологічного характеру, які можуть виразитися в значній напрузі, особливо при тривалій роботі. Віддзеркалення, включаючи віддзеркалення від вторинних джерел світла, повинне бути зведено до мінімуму. Для захисту від надмірної яскравості вікон можуть бути застосовані штори і екрани.

Залежно від орієнтації вікон рекомендується наступна фарбування стін і підлоги:

Вікна орієнтовані на південь: стіни зеленувато-блакитного або світло-блакитного кольору; підлога - зелений;

Вікна орієнтовані на північ: стіни світло-оранжевого або оранжево-жовтого кольору; підлога - червонувато-оранжевий;

Вікна орієнтовані на схід: стіни жовто-зеленого кольору; підлога зелена або червонувато-оранжевий;

Вікна орієнтовані на захід: стіни жовто-зеленого або голубувато-зеленого кольору; підлога зелена або червонувато-оранжевий.

У приміщеннях, де знаходиться комп'ютер, необхідно забезпечити наступні величини коефіцієнта віддзеркалення: для стелі: 60 ... 70%, для стін: 40 ... 50%, для підлоги: близько 30%. Для інших поверхонь і робочих меблів: 30 ... 40%.

3.2.2 Освітлення

Правильно спроектоване і виконане виробниче освітлення покращує умови зорової роботи, знижує стомлюваність, сприяє підвищенню продуктивності праці, благотворно впливає на виробниче середовище, надаючи позитивну психологічну дію на працюючого, підвищує безпеку праці і знижує травматизм.

Недостатність освітлення приводить до напруги зору, ослабляє увагу, приводить до настання передчасної стомленості. Надмірно яскраве освітлення викликає засліплення, роздратування і різь в очах. Неправильний напрямок світла

на робочому місці може створювати різкі тіні, відблиски, дезорієнтувати працюючого. Всі ці причини можуть призвести до нещасного випадку або профзахворювань, тому такий важливий правильний розрахунок освітленості.

Існує три види освітлення - природне, штучне і поєднане (природне і штучне разом.

Природне освітлення - освітлення приміщень денним світлом, що потрапляє через світлові прорізи в зовнішніх огороджуючих конструкціях приміщення. Природне освітлення характеризується тим, що змінюється в широких межах залежно від часу дня, пори року, характеру області і ряду інших чинників.

Штучне освітлення застосовується при роботі в темний час доби і вдень, коли не вдається забезпечити нормовані значення коефіцієнта природного освітлення (похмура погода, короткий світловий день). Освітлення, при якому недостатнє за нормами природне освітлення доповнюється штучним, називається змішаним освітленням.

Штучне освітлення підрозділяється на робоче, аварійне, евакуаційне, охоронне. Робоче освітлення, у свою чергу, може бути загальним або комбінованим. Загальне - освітлення, при якому світильники розміщуються у верхній зоні приміщення рівномірно, або, як розташоване устаткування. Комбіноване - освітлення, при якому до загального додається місцеве освітлення.

При виконанні робіт категорії високої зорової точності (найменший розмір об'єкту розрізнення 0,3 ... 0,5 мм) величина коефіцієнта природного освітлення (КЕО) повинна бути не нижче 1,5%, а при зоровій роботі середньої точності (найменший розмір об'єкту розрізнення 0,5 ... 1,0 мм) КЕО повинен бути не нижче 1,0%. В якості джерел штучного освітлення звичайно використовуються люмінесцентні лампи типа ЛБ, або ДРЛ, які попарно об'єднуються в світильники, які повинні розташовуватися рівномірно над робочими поверхнями.

Вимоги до освітленості в приміщеннях, де встановлені комп'ютери, наступні: при виконанні зорових робіт високої точності загальна освітленість повинна

складати 300лк, а комбінована - 750лк; аналогічні вимоги при виконанні робіт середньої точності - 200 і 300лк відповідно.

Крім того все поле зору повинне бути освітлено достатньо рівномірно - ця основна гігієнічна вимога. Іншими словами, ступінь освітлення приміщення і яскравість екрану комп'ютера повинні бути приблизно однаковими, оскільки яскраве світло в районі периферійного зору значно збільшує напруженість очей і, як наслідок, призводить до їх швидкої стомлюваності.

3.2.3 Параметри мікроклімату

Параметри мікроклімату можуть мінятися в широких межах, у той час як необхідною умовою життєдіяльності людини є підтримка постійності температури тіла завдяки терморегуляції, тобто здатності організму регулювати віддачу тепла в навколишнє середовище. Принцип нормування мікроклімату - створення оптимальних умов для теплообміну тіла людини з навколишнім середовищем.

Обчислювальна техніка є джерелом істотних тепловиділень, що може привести до підвищення температури і зниження відносної вологості в приміщенні. У приміщеннях, де встановлені комп'ютери, повинні дотримуватися певні параметри мікроклімату. У санітарних нормах СН-245-71 встановлені величини параметрів мікроклімату, що створюють комфортні умови. Ці норми встановлюються в залежності від пори року, характеру трудового процесу і характеру виробничого приміщення (див. табл. 3.1).

Об'єм приміщень, в яких розміщені працівники обчислювальних центрів, не повинен бути меншим 19,5 $м^3$ / людини з урахуванням максимального числа одночасно працюючих в зміну. Норми подачі свіжого повітря в приміщення, де розташовані комп'ютери, приведені в табл. 3.2.

Таблиця 3.1 Параметри мікроклімату для приміщень, де встановлені комп'ютери

Період року	Параметр мікроклімату	Величина
Холодний	Температура повітря в приміщенні Відносна вологість Швидкість руху повітря	22 ... 24 ° С 40 ... 60% до 0,1 м / с
Теплий	Температура повітря в приміщенні Відносна вологість Швидкість руху повітря	23 ... 25 ° С 40 ... 60% 0,1 ... 0,2 м / с

Таблиця 3.2 Норми подачі свіжого повітря в приміщення, де розташовані комп'ютери

Характеристика приміщення	Об'ємна витрата подається в приміщення свіжого повітря, м ^{3} / на одну людину в годину
Об'єм до 20м ^{3} на особу 20 ... 40м ^{3} на особу Більш 40м ^{3} на особу	Не менше 30 Не менше 20 Природна вентиляція

Для забезпечення комфортних умов використовуються як Організаційні методи (раціональна організація проведення робіт залежно від пори року і доби, чергування праці і відпочинку), так і технічні засоби (вентиляція, кондиціювання повітря, опалювальна система).

3.2.4 Шум і вібрація

Шум погіршує умови праці надаючи шкідливу дію на організм людини. Працюючі в умовах тривалої шумової дії випробовують дратівливість, головні болі, запаморочення, зниження пам'яті, підвищену стомлюваність, зниження апетиту, біль у вухах і т.д. Такі порушення в роботі ряду органів і систем організму

людини можуть викликати негативні зміни в емоційному стані людини аж до стресових. Під впливом шуму знижується концентрація уваги, порушуються фізіологічні функції, з'являється втома у зв'язку з підвищеними енергетичними витратами і нервово-психічним напруженням, погіршується мовна комутація. Все це знижує працездатність людини і її продуктивність, якість і безпеку праці. Тривала дія інтенсивного шуму [вище 80 дБ (А)] на слух людини приводить до його часткової або повної втрати.

У табл. 3.3 вказані граничні рівні звуку залежно від категорії тяжкості і напруженості праці, що є безпечними відносно збереження здоров'я і працездатності.

Таблиця 3.3 Граничні рівні звуку, дБ, на робочих місцях.

Категорія напруженості праці	Категорія важкості праці			
	I. Легка	II. Середня	III. Важка	IV. Дуже важка
I. Мало напружений	80	80	75	75
II. Помірно напружений	70	70	65	65
III. Напружений	60	60	-	-
IV. Дуже напружений	50	50	-	-

Рівень шуму на робочому місці математиків-програмістів і операторів відеоматеріалів не повинен перевищувати 50дБА, а в залах обробки інформації на обчислювальних машинах - 65дБА. Для зниження рівня шуму стіни і стеля приміщень, де встановлені комп'ютери, можуть бути облицьовані звукопоглинальними матеріалами. Рівень вібрації в приміщеннях обчислювальних центрів може бути понижений шляхом встановлення устаткування на спеціальні віброізолятори.

3.2.5 Ергономічні вимоги до робочого місця

Проектування робочих місць, забезпечених відеотерміналами, відноситься до числа важливих проблем ергономічного проектування в області обчислювальної техніки.

Робоче місце і взаємне розташування всіх його елементів повинне відповідати антропометричним, фізичним і психологічним вимогам. Велике значення має також характер роботи. Зокрема, при організації робочого місця програміста повинні бути дотримані наступні основні умови: оптимальне розміщення устаткування, що до складу робочого місця і достатній робочий простір, що дозволяє здійснювати всі необхідні рухи і переміщення.

Ергономічними аспектами проектування відеотермінальних робочих місць, зокрема, є: висота робочої поверхні, розміри простору для ніг, вимоги до розташування документів на робочому місці (наявність і розміри підставки для документів, можливість різного розміщення документів, відстань від очей користувача до екрану, документа, клавіатури і т.д.), характеристики робочого крісла, вимоги до поверхні робочого столу, регульованість елементів робочого місця.

Головними елементами робочого місця програміста є стіл і крісло. Основним робочим положенням є положення сидячи.

Робоча поза сидячи викликає мінімальне стомлення програміста. Раціональне планування робочого місця передбачає чіткий порядок і сталість розміщення предметів, засобів праці і документації. Те, що потрібно для виконання робіт частіше, розташоване в зоні легкої досяжності робочого простору.

Моторне поле - простір робочого місця, в якому можуть здійснюватися рухові дії людини.

Максимальна зона досяжності рук - це частина моторного поля робочого місця, обмеженого дугами, описуваними максимально витягнутими руками при русі їх у плечовому суглобі.

Оптимальна зона - частина моторного поля робочого місця, обмеженого дугами, описуваними передпліччями при русі в ліктьових суглобах з опорою в точці ліктя і з відносно нерухомим плечем.

Оптимальне розміщення предметів праці і документації в зонах досяжності:

- Дисплей розміщається в зоні а (у центрі);
- Системний блок розміщається в передбаченій ніші столу;
- Клавіатура - у зоні г / д;
- Маніпулятор «миша» - в зоні в справа;
- Сканер в зоні а / б (зліва);
- Принтер знаходиться в зоні а (праворуч);

Документація: необхідна при роботі - в зоні легкої досяжності долоні - в, а у висувних ящиках столу - література, невикористовувана постійно.

Для комфортної роботи стіл повинен задовольняти наступним умовам:

- висота столу повинна бути вибрана з урахуванням можливості сидіти вільно, в зручній позі, при необхідності спираючись на підлокітники;
- нижня частина столу повинна бути сконструйована так, щоб програміст міг зручно сидіти, не був змушений підбирати ноги;
- поверхняя столу повинна мати властивості, що виключають появу відблисків у поле зору програміста;
- конструкція столу повинна передбачати наявність висувних ящиків (не менше 3 для зберігання документації, лістингів, канцелярських приладдь).
- висота робочої поверхні рекомендується в межах 680-760мм. Висота поверхні, на яку встановлюється клавіатура, повинна бути близько 650мм.

Велике значення надається характеристикам робочого крісла. Так, рекомендована висота сидіння над рівнем підлоги перебуває в межах 420-550мм. Поверхня сидіння м'яка, передній край закруглений, а кут нахилу спинки - регульований.

Необхідно передбачати при проектуванні можливість різного розміщення документів: збоку від відеотерміналу, між монітором і клавіатурою і т.п. Крім того,

у випадках, коли відеотермінал має низьку якість зображення, наприклад помітні мелькання, відстань від очей до екрану роблять більше (біля 700мм), ніж відстань від ока до документа (300-450мм). Взагалі при високій якості зображення на відеотерміналі відстань від очей користувача до екрану, документа і клавіатури може бути рівним.

Положення екрану визначається:

- відстанню зчитування (0,6 ... 0,7 м);
- кутом зчитування, напрямком погляду на 20 ° нижче горизонталі до центру екрану, причому екран перпендикулярний цьому напрямку.

Повинна також передбачатися можливість регулювання екрану:

- по висоті +3 см;
- по нахилу від -10 ° до +20 ° щодо вертикалі;
- в лівому і правому напрямках.

Велике значення також надається правильній робочій позі користувача. При незручній робочій позі можуть з'явитися болі в м'язах, суглобах і сухожиллях. Вимоги до робочої пози користувача відеотермінала наступні:

- голова не повинна бути нахилена більш ніж на 20 °,
- плечі повинні бути розслаблені,
- лікті - під кутом 80 ° ... 100 °,
- передпліччя і кисті рук - в горизонтальному положенні.

Причина неправильної пози користувачів обумовлена наступними чинниками: немає хорошої підставки для документів, клавіатура знаходиться дуже високо, а документи - низько, нікуди покласти руки і кисті, недостатній простір для ніг.

З метою подолання вказаних недоліків даються загальні рекомендації: краще пересувна клавіатура; повинні бути передбачені спеціальні пристосування для регулювання висоти столу, клавіатури і екрану, а також підставка для рук.

Істотне значення для продуктивної і якісної роботи на комп'ютері мають розміри знаків, густину їх розміщення, контраст і співвідношення

яскравості символів і фону екрану. Якщо відстань від очей оператора до екрану дисплея складає 60 ... 80 см, то висота знака повинна бути не менше 3мм, оптимальне співвідношення ширини і висоти знака складає 3:4, а відстань між знаками - 15 ... 20% їх висоти. Співвідношення яскравості фону екрану і символів - від 1:2 до 1:15.

Під час користування комп'ютером медики радять встановлювати монітор на відстані 50-60 см від очей. Фахівці також вважають, що верхня частина відеодисплея повинна бути на рівні очей або трохи нижче. Коли людина дивиться прямо перед собою, його очі відкриваються ширше, ніж коли він дивиться вниз. За рахунок цього площа огляду значно збільшується, викликаючи обезводнення очей. До того ж якщо екран встановлений високо, а очі широко відкриті, порушується функція моргання. Це значить, що очі не закриваються повністю, не омиваються слізною рідиною, не одержують достатнього зволоження, що приводить до їх швидкої стомлюваності.

Створення сприятливих умов праці і правильне естетичне оформлення робочих місць на виробництві має велике значення як для полегшення праці, так і для підвищення його привабливості, позитивно впливає на продуктивність праці.

3.3 Розрахунок освітленості і рівня шуму

3.3.1 Розрахунок освітленості

Розрахунок освітленості робочого місця зводиться до вибору системи освітлення, визначенню необхідного числа світильників, їхнього типу і розміщення. Виходячи з цього, розрахуємо параметри штучного освітлення.

Зазвичай штучне освітлення виконується за допомогою електричних джерел світла двох видів: ламп накалювання і люмінесцентних ламп. Будемо використовувати люмінесцентні лампи, які порівняно з лампами розжарювання мають ряд істотних переваг:

- за спектральним складом світла вони близькі до денного, природного світла;

- володіють більш високим ККД (у 1,5-2 рази вище, ніж ККД ламп розжарювання);
- мають підвищену світловіддачею (в 3-4 рази вище, ніж у ламп розжарювання);
- більш тривалий термін служби.

Розрахунок освітлення проводиться для кімнати площею 15м ^{2,} ширина якої 5м, висота - 3 м. Скористаємося методом світлового потоку.

Для визначення кількості світильників визначимо світловий потік, падаючий на поверхню за формулою: $F = \frac{E*K*S*Z}{n}$, Де

F - розраховується світловий потік, Лм;

E - нормована мінімальна освітленість, Лк (визначається за таблицею). Роботу програміста, відповідно до цієї таблиці, можна віднести до розряду точних робіт, отже, мінімальна освітленість буде E = 300лк;

S - площа освітлюваного приміщення (у нашому випадку S = 15м $^{2);}$

Z - відношення середньої освітленості до мінімальної (звичайно приймається рівним 1,1 ... 1,2, нехай Z = 1,1);

K - коефіцієнт запасу, враховує зменшення світлового потоку лампи в результаті забруднення світильників у процесі експлуатації (його значення залежить від типу приміщення й характеру проведених у ньому робіт і в нашому випадку K = 1,5);

n - коефіцієнт використання, (виражається відношенням світлового потоку, що падає на розрахункову поверхню, до сумарного потоку всіх ламп і обчислюється в частках одиниці; залежить від характеристик світильника, розмірів приміщення, фарбування стін і стелі, які характеризуються коефіцієнтами відображення від стін ($P_{C)}$ і стелі ($P_{П)),}$ значення коефіцієнтів P_C і $P_П$ були зазначені вище: P_C = 40%, $P_П$ = 60%. Значення n визначимо по таблиці коефіцієнтів використання різних світильників. Для цього обчислимо індекс приміщення по формулі: : $I = \frac{S}{h(A+B)}$, Де

S - площа приміщення, S = 15 м $^{2;}$

h - розрахункова висота підвісу, h = 2.92 м;

A - ширина приміщення, A = 3 м;

B - довжина приміщення, B = 5 м.

Підставивши значення отримаємо: $I = \frac{15}{2.92(3+5)} = 0.64$

Знаючи індекс приміщення I, знаходимо n = 0,22

Підставимо всі значення у формулу для визначення світлового потоку F:

$$F = \frac{300 * 1.5 * 15 * 1.1}{0.22} =$$

Для освітлення вибираємо люмінесцентні лампи типу ЛБ40-1, світловий потік яких F = 4320 Лк.

Розрахуємо необхідну кількість ламп за формулою: $N = \frac{F}{F_л}$, Де

N - кількість ламп, що визначається;

F - світловий потік,

F = 33750 Лм;

$F_л$ - світловий потік лампи, $F_л$ = 4320 Лм.

$$N = \frac{33750}{4320} = 7,81$$

При виборі освітлювальних приладів використовуємо світильники типу ОД. Кожен світильник комплектується двома лампами.

3.3.1 Розрахунок рівня шуму

Одним з несприятливих факторів виробничого середовища в ІОЦ є високий рівень шуму, створюваний друкованими пристроями, обладнанням для кондиціонування повітря, вентиляторами систем охолодження в самих ЕОМ.

Для вирішення питань про необхідність і доцільність зниження шуму необхідно знати рівні шуму на робочому місці оператора.

Рівень шуму, що виникає від декількох некогерентних джерел, що працюють одночасно, підраховується на підставі принципу енергетичного підсумовування випромінювань окремих джерел:

$$L = 10lg \sum_{i=1}^{n} 10^{0.1*L_i}$$

де L_i - рівень звукового тиску і-го джерела шуму;

n - кількість джерел шуму.

Отримані результати розрахунку порівнюється з допустимим значенням рівня шуму для даного робочого місця. Якщо результати розрахунку вище допустимого значення рівня шуму, то необхідні спеціальні заходи щодо зниження шуму. До них відносяться: облицювання стін і стелі залу звукопоглинальними матеріалами, зниження шуму в джерелі, правильне планування обладнання і раціональна організація робочого місця оператора.

Рівні звукового тиску джерел шуму, що діють на оператора на його робочому місці представлені в табл. 3.4.

Таблиця 3.4 Рівні звукового тиску різних джерел.

Джерело шуму	Рівень шуму, дБ
Жорсткий диск	40
Вентилятор	45
Монітор	17
Клавіатура	10
Принтер	45
Сканер	42

Зазвичай робоче місце оператора оснащене наступним обладнанням: вінчестер в системному блоці, вентилятор (и) систем охолодження ПК, монітор, клавіатура, принтер і сканер.

Підставивши значення рівня звукового тиску для кожного виду обладнання у формулу, отримаємо:

$L_{\Sigma} = 10 \cdot \lg (10^{4} + 10^{4,5} + 10^{1,7} + 10^{1} + 10^{4,5} + 10^{4,2)} = 49,5$ дБ

Отримане значення не перевищує допустимий рівень шуму для робочого місця оператора, рівний 65 дБ (ГОСТ 12.1.003-83). І якщо врахувати, що навряд чи такі периферійні пристрої як сканер і принтер будуть використовуватися одночасно, то ця цифра буде ще нижчою. Крім того при роботі принтера безпосередню присутність оператора необов'язково, тому що принтер обладнаний механізмом автоподачі аркушів.

3.4 Заходи та засоби протипожежного захисту

Система протипожежного захисту - це сукупність організаційних заходів а також технічних засобів, спрямованих на запобігання впливу на людей небезпечних чинників пожежі та обмеження матеріальних збитків від неї.

Протипожежний захист об'єкта здійснюється за такими чотирма напрямками:

– обмеження розмірів та поширення пожежі:

1) розміщення будівель та споруд на території об'єкта із дотриманням протипожежних розривів та інших вимог пожежної безпеки;

2) дотримання обмежень стосовно кількості поверхів будівель та площі поверху;

3) правильне планування та розміщення виробничих цехів, приміщень, дільниць у межах будівлі;

4) розміщення легкозаймистих процесів та устаткування в ізольованих приміщеннях, відсіках, камерах;

5) вибір будівельних конструкцій необхідних ступенів вогнестійкості;

6) встановлювання протипожежних перешкод у будівлях, системах вентиляції, паливних та кабельних комунікаціях;

7) обмеження витікання та розтікання легкозаймистих та горючих рідин у разі пожежі;

8) облаштування систем автоматичної пожежної сигналізації та пожежогасіння;

– обмеження розвитку пожежі:

1) обмеження кількості горючих речовин, що одночасно знаходяться в приміщенні;

2) використання оздоблювальних будівельних та конструкційних матеріалів з нормативними показниками вибухонебезпечності та пожежонебезпечності;

3) аварійний викид горючих рідин та газів;

4) своєчасне звільнення приміщень від залишків горючих матеріалів;

5) застосування для легкозаймистих речовин спеціального устаткування із посиленим захистом від пошкоджень;

– забезпечення безпечної евакуації людей та майна:

1) вибір такого об'ємно-планувального та конструктивного виконання будівлі, щоб евакуація людей була завершена до настання гранично допустимих рівнів чинників пожежі;

2) застосування будівельних конструкцій будівель та споруд відповідних ступенів вогнестійкості, щоб вони зберігали несучі та огороджувальні функції протягом всього часу евакуації;

3) вибір відповідних засобів колективного та індивідуального захисту;

4) застосування аварійного вимкнення устаткування та комунікацій;

5) облаштування систем протидимового захисту, які запобігають задимленню шляхів евакуації;

6) влаштування необхідних шляхів евакуації (коридорів, сходових кліток, зовнішніх пожежних драбин), раціональне їх розміщення та належне утримання;

– створення умов для успішного гасіння пожежі:

1) встановлення в будівлях та приміщеннях установок пожежної автоматики;

2) забезпечення приміщень нормованою кількістю первинних засобів пожежогасіння;

3) облаштування та утримання в належному стані території підприємства, під'їздів до будівельних споруд, пожежних водоймищ, гідрантів.

Для приборкання пожежі в приміщенні обладнаному комп’ютерами та іншим обладнанням що знаходиться під напругою суворо заборонено використовувати повітряно-пінні, водні та аерозольні вогнегасники. Для гасіння електроприладів необхідно використовувати вуглекислотні та порошкові вогнегасники які придатні для гасіння пожеж класу Е, тобто електроприладів та електромереж, оскільки речовина що використовується в цих вогнегасника є діелектриком.

Вуглекислотні вогнегасники можна застосовувати майже до всіх видів пожеж. Принцип їхньої дії — витіснення кисню вуглекислим газом. Оскільки засобом гасіння у цих вогнегасниках є газ, їх не дуже ефективно використовувати

у великих приміщеннях, де є протяг. Але через те, що цей вогнегасник зовсім не забруднює середовище, він найкраще підходить для гасіння чутливих механізмів та електроприладів. Однак, при застосуванні вуглекислотного вогнегасника у невеликому приміщенні, після згасання полум'я, необхідно вийти і зачинити за собою двері, щоб не задихнутися вуглекислим газом.

Порошкові вогнегасники перешкоджають горінню хімічним шляхом і є майже універсальним протипожежним засобом. Порошок добре гасить пожежі не лише класу А і В, а й пожежі класу Е.

3.5 Висновки

У даному розділі кваліфікаційної роботи були викладені вимоги до робочого місця інженера - програміста. Створені умови повинні забезпечувати комфортну роботу. На підставі вивченої літератури з даної проблеми, були зазначені оптимальні розміри робочого столу і крісла, робочої поверхні, а також проведено вибір системи і розрахунок оптимального освітлення виробничого приміщення, а також розрахунок рівня шуму на робочому місці.

Дотримання умов, що визначають оптимальну організацію робочого місця інженера - програміста, дозволить зберегти гарну працездатність протягом усього робочого дня, підвищить як в кількісному, так і в якісному відносинах продуктивність праці програміста, що в свою чергу сприятиме якнайшвидшій розробці і налагодженню програмного продукту.

Висновки

Застосування сучасних технічних засобів дає можливість ефективно використовувати та миттєво отримувати інформаційний ресурс завдяки світовому інформаційному простору, що значно підвищує доступність до інформації.

Під час розробки пз було проведено аналіз потенційного апаратного забезпечення необхідного для коректної роботи розроблюваного програмного забезпечення. Для аналізу апаратного забезпечення необхідно встановити параметри необхідні для якісної роботи розроблюваного програмного модуля, а саме: критерії для опрацьовуємого матеріалу, апаратні вимоги для роботи програмного забезпечення та апаратні вимоги до технічного засобу збору матеріалу для аналізу.

При створені програмного засобу були враховані та виконані всі вимоги технічного завдання, а саме:

- візуальний інтерфейс користувача;
- реалізація пошуку потрібної інформації;

Програмний засіб є надзвичайно корисним та актуальним, адже він має такі переваги:

- простий та зрозумілий користувацький інтерфейс;
- низькі системні вимоги.

В подальшому програмний засіб може бути розширений та об’єднаний з іншими темами з програмування.

Перелік використаних джерел

1. ГОСТ 19.103–77. ЄСПД. Позначення програм і програмних документів [Текст]. – Введ. 1978-07-01. – М.: Государственный комитет СССР по стандартам.
2. ГОСТ 19.105–78. ЄСПД. Загальні вимоги до програмних документів, виконаних друкованим способом [Текст]. – Введ. 1980-01-01. – М.: Государственный комитет СССР по стандартам.
3. Безпека життєдіяльності. Підручник для вузів / С. В. Бєлов, А. В. Ільницька, А. Ф. Козьяков и др. - М.: Вища школа, 2005. - 448 с.
4. Гандзюк М. Основи охорони праці : Підручник для студ. вуз./ Ми-хайло Гандзюк, Євген Желібо, Модест Халімовський, За ред. Михайла Гандзюка. - 2-е вид.. -К.: Каравела, 2004. -405 с.
5. Грищук М. Основи охорони праці : Підручник/ Микола Грищук,; М-во освіти і науки України, ун-т "Острозька академія". -К.: Кондор, 2005. - 238 с.
6. Бьёрн Страуструп. Язык программирования C++ = The C++ Programming Language / Пер. з англ. — 3-е изд. — СПб.; М.: Невский диалект — Бином, 1999. — 991 с. — ISBN 5-7940-0031.
7. Herbert Schildt. C++ The Complete Reference Third Edition. — Osborne McGraw-Hill, 1998. — ISBN 978-0-07-882476-0.
8. Джон Скит. C# для профессионалов: тонкости программирования, 3-е издание — М.: «Вильямс», 2014. — 608 с. — ISBN 978-5-8459-1909-0.
9. Stewart, Bill (2000-01-07). History of the C Programming Language. Living Internet. Архів оригіналу за 2013-06-22.
10. Jeff Prosise: «Windows Forms: Современная модель программирования для создания GUI приложений».
11. Вибір між C++ і C# курс [Електронний ресурс] – Режим доступу до ресурсу: https://habr.com/post/262461.

Додаток А

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Water
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
      }
      private void resurs_Click(object sender, EventArgs e)
      {
         using (OpenFileDialog ofd = new OpenFileDialog())
         {
            if (ofd.ShowDialog() == DialogResult.Cancel) checkimage = false;
            else
            {
               resurs.Image = new Bitmap(ofd.FileName);
               checkimage = true;
               checksize((Bitmap)resurs.Image);
            }
         }
      }
      public bool checkimage = false;
      public bool checkimagesize = false;
      public bool triger = false;
      private void button1_Click(object sender, EventArgs e)
      {
         if (checkimage == false)
         {
            masage.Text += "Не обрано опрацьовуэме зображення\n";
         }
         else if (checkimagesize == false)
            masage.Text += "Зображення недостатньо якісне\n";
         else if (checkimage != false && checkimagesize != false)
```

```
        {
            masage.Text = "-----------------------------------------------------------------\n";
            masage.Text += "Працюємо\n";
            Bitmap rgb = work((Bitmap)resurs.Image);
            result.Image = rgb;
            triger = true;
            masage.Text += "Готово\n";
            masage.Text += "-----------------------------------------------------------------\n";
        }
        else
            masage.Text = "Неопізнана помилка\n";
    }
    private void checksize(Bitmap source)
    {
        if (source.Width < 800 && source.Height < 600)
            checkimagesize = false;
        else checkimagesize = true;
    }
    public Bitmap work(Bitmap source)
    {
        int controlr, controlg, controlb;
        Bitmap result = new Bitmap(source.Width, source.Height);
        Color col = source.GetPixel((source.Width / 2), (source.Height / 2));
        controlr = col.R;
        controlg = col.G;
        controlb = col.B;
        if (controlr >= 24 && controlr <= 35 && controlg >= 30 && controlg <= 50
&& controlb >= 30 && controlb <= 65)
            result = Getthird(source);
        else if (controlr >= 0 && controlr <= 110 && controlg >= 30 && controlg <=
120 && controlb >= 45 && controlb <= 170)
            result = Getsecond(source);
        else if (controlr >= 0 && controlr <= 150 && controlg >= 70 && controlg <=
180 && controlb >= 100 && controlb <= 205)
            result = Getfirst(source);
        else
            masage.Text = "Помилка з зорбраженням\n";
        return result;
    }
    public Bitmap Getfirst(Bitmap source)
    {
        int[] persent = new int[10];
        Bitmap result = new Bitmap(source.Width, source.Height);
```

```
        for (int i = 0; i < source.Width; i++)
        {
            for (int j = 0; j < source.Height; j++)
            {
                Color color = source.GetPixel(i, j);

                if (color.R >= 0 && color.R <= 150 && color.G >= 70 && color.G <= 180
&& color.B >= 100 && color.B <= 205)
                {
                    if (color.R <= 15 && color.G <= 27 && color.B <= 48)
                    {
                        result.SetPixel(i, j, Color.FromArgb(color.A, 255, 0, 0));
                        persent[0]++;
                    }
                    else if (color.R < 30 && color.G < 44 && color.B < 66)
                    {
                        result.SetPixel(i, j, Color.FromArgb(color.A, 255, 51, 0));
                        persent[1]++;
                    }
                    else if (color.R < 45 && color.G < 61 && color.B < 84)
                    {
                        result.SetPixel(i, j, Color.FromArgb(color.A, 255, 102, 0));
                        persent[2]++;
                    }
                    else if (color.R < 60 && color.G < 78 && color.B < 102)
                    {
                        result.SetPixel(i, j, Color.FromArgb(color.A, 255, 153, 0));
                        persent[3]++;
                    }
                    else if (color.R < 75 && color.G < 95 && color.B < 120)
                    {
                        result.SetPixel(i, j, Color.FromArgb(color.A, 255, 204, 0));
                        persent[4]++;
                    }
                    else if (color.R < 90 && color.G < 112 && color.B < 138)
                    {
                        result.SetPixel(i, j, Color.FromArgb(color.A, 204, 255, 0));
                        persent[5]++;
                    }
                    else if (color.R < 105 && color.G < 129 && color.B < 156)
                    {
                        result.SetPixel(i, j, Color.FromArgb(color.A, 153, 255, 0));
                        persent[6]++;
```

```
                }
                else if (color.R < 120 && color.G < 146 && color.B < 174)
                {
                    result.SetPixel(i, j, Color.FromArgb(color.A, 102, 255, 0));
                    persent[7]++;
                }
                else if (color.R < 135 && color.G < 163 && color.B < 192)
                {
                    result.SetPixel(i, j, Color.FromArgb(color.A, 51, 255, 0));
                    persent[8]++;
                }
                else if (color.R <= 150 && color.G <= 180 && color.B <= 210)
                {
                    result.SetPixel(i, j, Color.FromArgb(color.A, 0, 255, 0));
                    persent[9]++;
                }
                else
                    result.SetPixel(i, j, Color.FromArgb(color.A, color.R, color.G,
color.B));
            }
            else
                result.SetPixel(i, j, Color.FromArgb(color.A, color.R, color.G, color.B));
        }
    }
    Masager(persent, source.Width, source.Height);
    return result;
}
public Bitmap Getsecond(Bitmap source)
{
    int[] persent = new int[10];
    Bitmap result = new Bitmap(source.Width, source.Height);
    for (int i = 0; i < source.Width; i++)
    {
        for (int j = 0; j < source.Height; j++)
        {
            Color color = source.GetPixel(i, j);
            if (color.R >= 0 && color.R <= 110 && color.G >= 30 && color.G <= 120
&& color.B >= 45 && color.B <= 170)
            {
                if (color.R <= 15 && color.G <= 27 && color.B <= 48)
                {
                    result.SetPixel(i, j, Color.FromArgb(color.A, 255, 0, 0));
                    persent[0]++;
```

```
}
else if (color.R < 30 && color.G < 44 && color.B < 66)
{
   result.SetPixel(i, j, Color.FromArgb(color.A, 255, 51, 0));
   persent[1]++;
}
else if (color.R < 45 && color.G < 61 && color.B < 84)
{
   result.SetPixel(i, j, Color.FromArgb(color.A, 255, 102, 0));
   persent[2]++;
}
else if (color.R < 60 && color.G < 78 && color.B < 102)
{
   result.SetPixel(i, j, Color.FromArgb(color.A, 255, 153, 0));
   persent[3]++;
}
else if (color.R < 75 && color.G < 95 && color.B < 120)
{
   result.SetPixel(i, j, Color.FromArgb(color.A, 255, 204, 0));
   persent[4]++;
}
else if (color.R < 90 && color.G < 112 && color.B < 138)
{
   result.SetPixel(i, j, Color.FromArgb(color.A, 204, 255, 0));
   persent[5]++;
}
else if (color.R < 105 && color.G < 129 && color.B < 156)
{
   result.SetPixel(i, j, Color.FromArgb(color.A, 153, 255, 0));
   persent[6]++;
}
else if (color.R < 120 && color.G < 146 && color.B < 174)
{
   result.SetPixel(i, j, Color.FromArgb(color.A, 102, 255, 0));
   persent[7]++;
}
else if (color.R < 135 && color.G < 163 && color.B < 192)
{
   result.SetPixel(i, j, Color.FromArgb(color.A, 51, 255, 0));
   persent[8]++;
}
else if (color.R <= 150 && color.G <= 180 && color.B <= 210)
{
```

```
                    result.SetPixel(i, j, Color.FromArgb(color.A, 0, 255, 0));
                    persent[9]++;
                }
                else
                    result.SetPixel(i, j, Color.FromArgb(color.A, color.R, color.G,
color.B));
            }
            else
                result.SetPixel(i, j, Color.FromArgb(color.A, color.R, color.G, color.B));
        }
    }
    Masager(persent, source.Width, source.Height);
    return result;
}
public Bitmap Getthird(Bitmap source)
{
    int[] persent = new int[10];
    Bitmap result = new Bitmap(source.Width, source.Height);
    for (int i = 0; i < source.Width; i++)
    {
        for (int j = 0; j < source.Height; j++)
        {
            Color color = source.GetPixel(i, j);
            if (color.R >= 24 && color.R <= 35 && color.G >= 30 && color.G <= 50
&& color.B >= 30 && color.B <= 65)
            {
                if (color.R <= 15 && color.G <= 27 && color.B <= 48)
                {
                    result.SetPixel(i, j, Color.FromArgb(color.A, 255, 0, 0));
                    persent[0]++;
                }
                else if (color.R < 30 && color.G < 44 && color.B < 66)
                {
                    result.SetPixel(i, j, Color.FromArgb(color.A, 255, 51, 0));
                    persent[1]++;
                }
                else if (color.R < 45 && color.G < 61 && color.B < 84)
                {
                    result.SetPixel(i, j, Color.FromArgb(color.A, 255, 102, 0));
                    persent[2]++;
                }
                else if (color.R < 60 && color.G < 78 && color.B < 102)
                {
```

```
                    result.SetPixel(i, j, Color.FromArgb(color.A, 255, 153, 0));
                    persent[3]++;
                }
                else if (color.R < 75 && color.G < 95 && color.B < 120)
                {
                    result.SetPixel(i, j, Color.FromArgb(color.A, 255, 204, 0));
                    persent[4]++;
                }
                else if (color.R < 90 && color.G < 112 && color.B < 138)
                {
                    result.SetPixel(i, j, Color.FromArgb(color.A, 204, 255, 0));
                    persent[5]++;
                }
                else if (color.R < 105 && color.G < 129 && color.B < 156)
                {
                    result.SetPixel(i, j, Color.FromArgb(color.A, 153, 255, 0));
                    persent[6]++;
                }
                else if (color.R < 120 && color.G < 146 && color.B < 174)
                {
                    result.SetPixel(i, j, Color.FromArgb(color.A, 102, 255, 0));
                    persent[7]++;
                }
                else if (color.R < 135 && color.G < 163 && color.B < 192)
                {
                    result.SetPixel(i, j, Color.FromArgb(color.A, 51, 255, 0));
                    persent[8]++;
                }
                else if (color.R <= 150 && color.G <= 180 && color.B <= 210)
                {
                    result.SetPixel(i, j, Color.FromArgb(color.A, 0, 255, 0));
                    persent[9]++;
                }
                else
                    result.SetPixel(i, j, Color.FromArgb(color.A, color.R, color.G,
color.B));
            }
            else
                result.SetPixel(i, j, Color.FromArgb(color.A, color.R, color.G, color.B));
        }
    }
    Masager(persent, source.Width, source.Height);
    return result;
```

```
        }
        public void Masager(int[] pixel, int w, int h)
        {
            float summ, voda;
            summ = pixel[0] + pixel[1] + pixel[2] + pixel[3] + pixel[4] + pixel[5] + pixel[6]
+ pixel[7] + pixel[8] + pixel[9];
            float[] persent = new float[10];
            voda = summ/(w*h)*100;
            for (int i = 0; i < 10; i++)
            {
               persent[i] = (pixel[i] / summ) * 100;
            }
            masage.Text += "----------------------------------------------------------------------------
--\n";
            masage.Text += "Відносна площа водойми до всієї зображеної ";
            masage.Text += voda;
            masage.Text += "%\n";
            masage.Text += "----------------------------------------------------------------------------
--\n";
            masage.Text += "Відносні площі забрудненої води до всього водойому
розподілені за рівнями забруднення\n";
            masage.Text += "----------------------------------------------------------------------------
--\n";
            masage.Text += "Критерії рівнів забруднення за 10-бальною шкалою\n";
            masage.Text += "Де 1 - критичне забруднення\n";
            masage.Text += "10 - ідеально чисто\n";
            masage.Text += "----------------------------------------------------------------------------
--\n";
            masage.Text += "1  ";
            masage.Text += persent[0];
            masage.Text += "%\n";
            masage.Text += "2  ";
            masage.Text += persent[1];
            masage.Text += "%\n";
            masage.Text += "3  ";
            masage.Text += persent[2];
            masage.Text += "%\n";
            masage.Text += "4  ";
            masage.Text += persent[3];
            masage.Text += "%\n";
            masage.Text += "5  ";
            masage.Text += persent[4];
            masage.Text += "%\n";
```

```
            masage.Text += "6  ";
            masage.Text += persent[5];
            masage.Text += "%\n";
            masage.Text += "7  ";
            masage.Text += persent[6];
            masage.Text += "%\n";
            masage.Text += "8  ";
            masage.Text += persent[7];
            masage.Text += "%\n";
            masage.Text += "9  ";
            masage.Text += persent[8];
            masage.Text += "%\n";
            masage.Text += "10  ";
            masage.Text += persent[9];
            masage.Text += "%\n";
            masage.Text += "-------------------------------------------------------------------------
--\n";
            masage.Text += "Дані розрахунки несуть рекомендаційний характер.\n";
            masage.Text += "\n";
            masage.Text += "Розрагунки мають достатню точність для того, щоб
виявити забруднення та приблизний його розмір.\n";
            masage.Text += "\n";
            masage.Text += "У випадку виявлення високого рівня забруднення слід
визначити характер забруднення, а саме викликане воно наявністю сторонніх
предметів чи ні.\n";
            masage.Text += "\n";
            masage.Text += "У випадку коли причина забруднення не є сторонні
предмети слід звернутись до спеціальних служб для проведення більш детального
аналізу водоймища за допомогою спеціального обладнання. \n";
            masage.Text += "-------------------------------------------------------------------------
--\n";
        }
    }
}

namespace Water
{
    partial class Form1
    {
        /// <summary>
        /// Обязательная переменная конструктора.
        /// </summary>
        private System.ComponentModel.IContainer components = null;
```

```
        /// <summary>
        /// Освободить все используемые ресурсы.
        /// </summary>
        /// <param name="disposing">истинно, если управляемый ресурс должен быть удален; иначе ложно.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Код, автоматически созданный конструктором форм Windows

        /// <summary>
        /// Требуемый метод для поддержки конструктора — не изменяйте
        /// содержимое этого метода с помощью редактора кода.
        /// </summary>
        private void InitializeComponent()
        {
            this.resurs = new System.Windows.Forms.PictureBox();
            this.label1 = new System.Windows.Forms.Label();
            this.button1 = new System.Windows.Forms.Button();
            this.masage = new System.Windows.Forms.RichTextBox();
            this.label4 = new System.Windows.Forms.Label();
            this.result = new System.Windows.Forms.PictureBox();
            ((System.ComponentModel.ISupportInitialize)(this.resurs)).BeginInit();
            ((System.ComponentModel.ISupportInitialize)(this.result)).BeginInit();
            this.SuspendLayout();
            //
            // resurs
            //
            this.resurs.BorderStyle = System.Windows.Forms.BorderStyle.FixedSingle;
            this.resurs.Location = new System.Drawing.Point(0, 15);
            this.resurs.Name = "resurs";
            this.resurs.Size = new System.Drawing.Size(320, 240);
            this.resurs.SizeMode = System.Windows.Forms.PictureBoxSizeMode.Zoom;
            this.resurs.TabIndex = 0;
            this.resurs.TabStop = false;
            this.resurs.Click += new System.EventHandler(this.resurs_Click);
            //
```

```
// label1
//
this.label1.AutoSize = true;
this.label1.Location = new System.Drawing.Point(0, 0);
this.label1.Name = "label1";
this.label1.Size = new System.Drawing.Size(110, 13);
this.label1.TabIndex = 2;
this.label1.Text = "Обране зображення";
//
// button1
//
this.button1.Location = new System.Drawing.Point(0, 670);
this.button1.Name = "button1";
this.button1.Size = new System.Drawing.Size(320, 28);
this.button1.TabIndex = 13;
this.button1.Text = "Опрацювати";
this.button1.UseVisualStyleBackColor = true;
this.button1.Click += new System.EventHandler(this.button1_Click);
//
// masage
//
this.masage.Location = new System.Drawing.Point(0, 274);
this.masage.Name = "masage";
this.masage.Size = new System.Drawing.Size(320, 391);
this.masage.TabIndex = 14;
this.masage.Text = "";
//
// label4
//
this.label4.AutoSize = true;
this.label4.Location = new System.Drawing.Point(-3, 258);
this.label4.Name = "label4";
this.label4.Size = new System.Drawing.Size(58, 13);
this.label4.TabIndex = 22;
this.label4.Text = "Історія дій";
//
// result
//
this.result.BorderStyle = System.Windows.Forms.BorderStyle.FixedSingle;
this.result.Location = new System.Drawing.Point(335, 0);
this.result.Name = "result";
this.result.Size = new System.Drawing.Size(1014, 698);
this.result.SizeMode = System.Windows.Forms.PictureBoxSizeMode.Zoom;
```

```
            this.result.TabIndex = 23;
            this.result.TabStop = false;
            //
            // Form1
            //
            this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
            this.ClientSize = new System.Drawing.Size(1350, 697);
            this.Controls.Add(this.result);
            this.Controls.Add(this.label4);
            this.Controls.Add(this.masage);
            this.Controls.Add(this.button1);
            this.Controls.Add(this.label1);
            this.Controls.Add(this.resurs);
            this.Name = "Form1";
            this.Text = "Water";
            ((System.ComponentModel.ISupportInitialize)(this.resurs)).EndInit();
            ((System.ComponentModel.ISupportInitialize)(this.result)).EndInit();
            this.ResumeLayout(false);
            this.PerformLayout();

        }

        #endregion

        private System.Windows.Forms.PictureBox resurs;
        private System.Windows.Forms.Label label1;
        private System.Windows.Forms.PictureBox Water1;
        private System.Windows.Forms.PictureBox Water2;
        private System.Windows.Forms.PictureBox Water3;
        private System.Windows.Forms.PictureBox Water4;
        private System.Windows.Forms.Button button1;
        private System.Windows.Forms.RichTextBox masage;
        private System.Windows.Forms.Button button2;
        private System.Windows.Forms.Button button3;
        private System.Windows.Forms.Button button4;
        private System.Windows.Forms.Button button5;
        private System.Windows.Forms.Label label4;
        private System.Windows.Forms.PictureBox result;
    }
}
```