Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)

Кафедра Системотехніки
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

Другий (магістерський)
(рівень вищої освіти)

Розробка та дослідження інформаційної технології яскравісного вирівнювання зображень
(тема)

Виконав:
студент 2 курсу, групи ІТПм-21-1
Захарова К. О.
(прізвище, ініціали)

Спеціальність 122 – Комп'ютерні науки
(код і повна назва спеціальності)
Тип програми освітньо-професійна
Освітня програма Інформаційні технології проектування
(повна назва освітньої програми)

Керівник проф. Саваневич В.Є.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри системотехніки _____ Гребеннік І. В.
(підпис) (прізвище, ініціали)

2022 р.

Я як студентка ХНУРЕ розумію і підтримую політику закладу із академічної доброчесності. Я не надавала і не одержувала недозволену допомогу під час підготовки кваліфікаційної роботи. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

18.12.2022  Захарова К.О.

Кваліфікаційна робота не містить відомостей заборонених до відкритого опублікування.

Кваліфікаційна робота виконана у відповідності до стандартів, що діють в Україні.

Попередній захист проведено 19 грудня 2022 р.

Керівник кваліфікаційної роботи  проф. Саваневич В.Є.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
Кафедра Системотехніки
Освітньо-кваліфікаційний рівень Другий (магістрський)
Спеціальність 122 – Комп'ютерні науки
(код і назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри СТ Гребеннік І. В.
(підпис)
« » 2022 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ (ПРОЕКТ)

студентові Захаровій Катерині Олександрівні
(прізвище, ім'я, по батькові)

1. Тема роботи (проекту) Розробка та дослідження інформаційної технології яскравісного вирівнювання зображень
затверджена наказом по університету від « » 2022 р. №

2. Термін подання студентом роботи до екзаменаційної комісії

3. Вихідні дані до роботи Функції системи: надання зображення з подавленими шумами та вирівняною яскравістю Перелік використовуваних програмних засобів: ОС Microsoft Windows v.7 або вище, додатковий інструментарій OpenCV, середовище розробки PyCharm.

4. Перелік питань, що потрібно опрацювати в роботі 4.1 Вступ 4.2 Аналіз предметної області 4.2.1 Аналіз області обробки зображень 4.2.2 Аналіз попередньої обробки зображень 4.2.3 Медіанний фільтр 4.2.4 Аналіз існуючих методів вирівнювання яскравості і контрасності зображень 4.2.5 Аналіз яскравісного вирівнювання кадрів інверсним медіанним фільтром 4.2.6 Аналіз розробки програмного забезпечення 4.3 Вибір інструментарію розробки 4.3.1 Огляд мови програмування 4.3.2 Огляд середовища розробки 4.3.3 Огляд додаткового інструментарію 4.3 Проектування і розробка системи 4.3.1 Побудова контексної діаграми IDEF0 4.3.2 Аналіз варіантів використання 4.3.3 Побудова діаграми діяльності системи 4.3.4 Проектування внутрішньої будови системи за допомогою діаграми класів 4.3.5 Розробка інтерфейсу користувача 4.3.6 Тестування розробленого програмного забезпечення 4.4 Висновки 4.5 Перелік джерел посилання 4.6 Додаток А Текст програми

5 Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій 5.1 Діаграма прецедентів(1 аркуш формату А4). 5.2 Діаграма класів(1 аркуш формату А4). 5.3 Контекстна діаграма А-0(1 аркуш формату А4). 5.4 Діаграма діяльності (1 аркуш формату А4). 5.5Графічний інтерфейс програми (1 аркуш формату А4).

6. Консультанти розділів роботи

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата
	проф. Саваневич В.Є.		
	проф. Саваневич В.Є.		

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи (проекту)	Термін виконання етапів роботи	Примітка
1	Отримання завдання атестаційної роботи	21.11.22	виконано
2	Аналіз завдання, літератури та аналогів з теми атестаційної роботи	22.11 – 25.11.22	виконано
3	Вибір засобів для розробки технічних вимог до програми	26.11 – 27.11.22	виконано
4	Структурне проектування	28.11 – 03.12.22	виконано
5	Вибір середовища розробки програми	04.11 – 05.11.22	виконано
6	Розробка програми	06.11 – 14.11.22	виконано
7	Тестування програми	14.11 – 15.11.22	виконано
8	Оформлення пояснювальної записки та програмної документації	15.11 – 17.11.22	виконано
9	Оформлення графічної частини та презентаційних матеріалів комп'ютерного захисту	18.11.22	виконано
10	Представлення на рецензування	18.11.22	виконано
11	Представлення атестаційної роботи в ДЕК	21.11.22	виконано

Дата видачі завдання _____ _____ 2022 р.

Студент _____ Захарова К.О.
(підпис)

Керівник роботи _____ проф. Саваневич В.Є.
(підпис)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 66 с., 1 табл., 7 рис., 30 джерел посилання.

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, ВИРІВНЮВАННЯ ЯСКРАВОСТІ, CASE-ЗАСОБИ, ІНТЕРФЕЙС, КОРИСТУВАЧ, АВТОМАТИЗАЦІЯ, ШУМИ

Об’єктом дослідження – програмне забезпечення з вирівнювання яскравості зображень.

Предмет дослідження — процес цифрової обробки зображень, що включає в себе подавлення шумів та вирівнювання яскравості зображень.

Мета – дослідження методів цифрової обробки зображень, що включає в себе подавлення шумів та вирівнювання яскравості зображень.

В ході написання кваліфікаційної роботи було проведено аналіз предметної області цифрової обробки зображень; проведено огляд поняття прикладного ПЗ; проведено огляд поняття попереднбої обробки зображень, вирівнювання яскравості, контрасності; проведено аналіз існуючих методів та засобівобробки зображень та обрано методи, що будуть використовуватися в роботі, побудовані діаграми для проектування системи та практична реалізація системи.

Методами дослідження обрані методи медіанної фільтрації для подавлення шумів та інверсивний медіанний фільтр, як алгоритм вирівнювання освітленості та чіткості зображень.

Результати роботи – розроблене програмне забезпечення, що дозвоняє користувачу сисеми зробити подавлення шумів зображення та яскравісне вирівнювання.

ABSTRACT

Explanatory note contains 66 sheets, 1 tables, 30 sources, 7 images, 9 graphic materials.

SOFTWARE, BRIGHTNESS LEVELING, CASE TOOLS, INTERFACE, USER, AUTOMATION, NOISE

The object of the research is software for leveling the brightness of images.

The subject of research is the process of digital image processing, which includes noise reduction and image brightness equalization.

The goal is to study the methods of digital image processing, which includes noise reduction and image brightness equalization.

In the course of writing the qualification work, an analysis of the subject area of digital image processing was carried out; the concept of applied software was reviewed; an overview of the concepts of image preprocessing, brightness leveling, and contrast was carried out; an analysis of existing methods and means of image processing was carried out and the methods to be used in the work were chosen, diagrams were built for system design and practical implementation of the system.

Median filtering methods for noise suppression and inverse median filter as an algorithm for leveling illumination and clarity of images are chosen as research methods.

The results of the work are developed software that calls the system user to perform image noise reduction and brightness leveling.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

ІС – інформаційна система;

ПЗ – програмне забезпечення;

IDEF0 – стандарт методології функціонального моделювання;

UML – Unified Modeling Language;

ВСТУП

В сучасному світі люди все частіше зіштовхуються з необхідністю обробляти великі обсяги даних, робити з ними певні перетворення, тощо.

На хвилі загального розвитку технологій розпізнавання об’єктів, облич, тощо, велику популярність отримало явище обробки зображень найрізноманітнішого характеру, починаючи від обрізання і масок, закінчуючи вирівнюванням діаграм яскравості і контрастності.

Виходячи з популярності явища обробки зображень, об’єктом роботи є програмне забезпечення, що буде обробляти зображення.

Предмет дослідження — процес цифрової обробки зображень, що включає в себе подавлення шумів та вирівнювання яскравості зображень.

Мета роботи — створення системи для автоматичного подавлення шумів та вирівнювання яскравості зображення.

Для виконання поставленої мети слід виконати наступні завдання:

– провести аналіз предметної області обробки зображень;

– провести огляд поняття вирівнювання яскравості;

– обрати методи обробки зображень для подальшого засосування в програмі;

– провести аналіз методів та засобів розробки ПЗ;

– обрати мову програмування;

– обрати додаткові інструменти;

– обрати середовище розробки;

– проаналізувати концепцію системи та провести її проектування;

– провести аналіз варіантів використання;

– розробити графічний інтерфейс користувача;

– провести тестування системи.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз області обробки зображень

Цифрова обробка зображень є бурхливо розвивається областю науки. У сучасному світі з'являється все більша потреба в використання інформації, що має фото та відео формат. Інформація даного виду, а також технології, що дозволяють зберігати, редагувати та обробляти її, використовуються у повсюдних для нас сферах – системи відео спостереження, реклама, медичні прилади, аналіз металографічних зображень металів та сплавів тощо.

За даними американського статистичного економічного агентства Marketline Companies на 2016-й рік обсяг програмно-апаратних засобів, що продаються, пов'язаних із захопленням, обробкою та зберіганням фото та відео зображень збільшується щорічно на 20 відсотків, що призводить до пропорційному щорічному приросту фото та відео зображень. Тому досліджувана предметна область та тема є актуальними.

1.2 Аналіз попередньої обробки зображень

Процес обробки зображень складається з кількох етапів. Самим важливим є попередня обробка зображень. Цей етап відповідає за фільтрацію шумів, перешкод та інших недоліків на зображенні.

Завдання попередньої обробки зображень такі:

– придушення шумів;

– корекція кольорів;

– зміна діапазону яскравостей;

– корекція геометричних спотворень;

– зміна різкості.

Головним завданням попередньої обробки зображення є покращення якості зображення. Методи попередньої обробки різноманітні та залежать від завдання досліджень та від того, яким шляхом отримано вихідне зображення.

Шумопридушення зображень, в основному, служить для покращення візуального сприйняття. Але крім цього, шумозаглушення застосовують і в практичних цілях, наприклад у медицині, збільшуючи чіткість зображення на рентгенівських знімках.

У багатоканальному зображенні шум може мати різну інтенсивність для різних каналів. Внаслідок цього шум може мати різні відтінки.

Найбільш помітний шум на однотонних ділянках. Чим темніша ділянка, тим краще його видно [3].

Джерелами шумів можуть бути різні речі:

– умови зйомки;

– спотворення даних при пошкодженні інформації на носії;

– шум перенесення заряду;

– шум квантування АЦП;

– тепловий шум матриці;

– перешкоди під час передачі аналоговими каналами лінії передачі.

Шуми також піддаються розподілу на види. За типом спотворення шуми можна розділити на:

– адитивний шум, що виникає через апаратуру;

– мультиплікативний шум, що виникає через можливі зміни параметрів каналу передачі;

– імпульсний шум, що виникає через проблеми з програмним забезпеченням;

Якщо скористатися візуальним підходом, шуми можна класифікувати як:

– білий шум — пікселі випадкові та не кореловані один з одним;

– імпульсний шум — характеризується заміною значень деяких пікселів на зображенні на випадкові чи фіксовані значення;

– «биті пікселі» — пікселі або області пікселів, що мають випадкові

значення, найчастіше пов'язані з дефектом апаратури;

– Яскраві плями — характерні для аналогового сигналу;

– Подряпини, що виникають через фізичне пошкодження плівки або недоліків на об'єктиві [4].

1.3 Медіанний фільтр

Імпульсні шуми зазвичай виникають від псевдо випадкових подій. Наприклад, поруч із вашим пристроєм може перемикатися двосмуговий радіоприймач або може статися якийсь статичний розряд. Щоразу коли це відбувається, вхідний сигнал може тимчасово спотворюватися.

Наприклад, в результаті аналогово-цифрового перетворення ми отримуємо такий ряд значень: 385, 389, 912, 388, 387. Значення 912 імовірно аномальне і його потрібно відхилити. Якщо ви спробуєте використовувати класичний лінійний фільтр, то помітите, що значення 912 значний вплив на вихідний результат.

Найкращим рішенням у цьому випадку буде використання медіанного фільтра. Він, якраз, підходить для усунення різного роду дрібних розосереджених вкраплень, проте не є ідеальним інструментом т.к. пропускає трохи більші області зосереджених перешкод.

Медіанний фільтр є плаваючим вікном, розмірністю 3x3 пікселя стосовно зображення нижче. На вхід він набуває 9 значень (пікселів), а на вихід видає одне. Ідея медіанного фільтра проста. Він вибирає із групи вхідних значень середнє та видає на вихід. Зазвичай, група має непарну кількість значень, тому проблеми з вибором не виникає.

Працює медіанний фільтр так: сортує вхідні дані (пікселі) у порядку зростання та видає серединний результат (медіану).

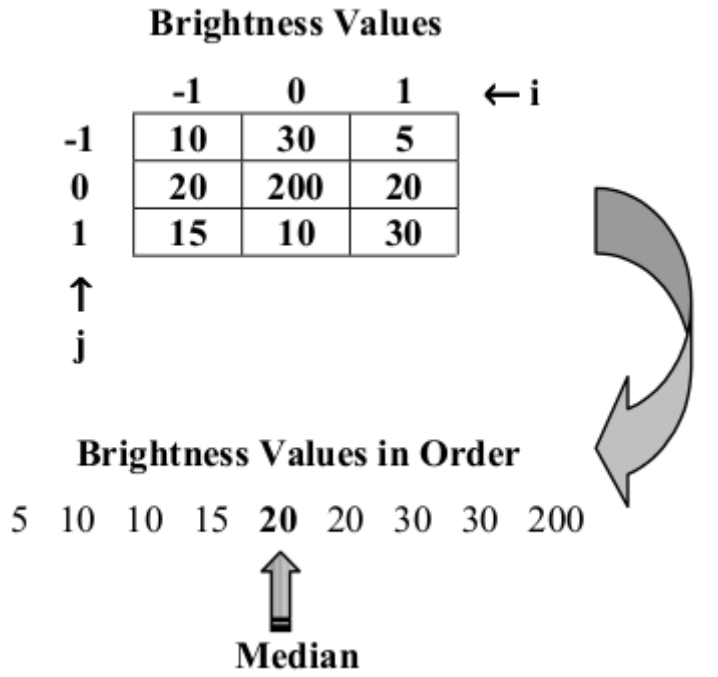

Рисунок 1.1 – Метод медіанної фільтрації

Незважаючи на очевидність такого підходу, медіанні фільтри використовуються у системах, що вбудовуються, рідко, що може бути пов'язано з нестачею знань про їхнє існування та труднощами з реалізацією.

Через простоту та лаконічність медіанної фільтрації було вирішено взяти цей метод за основний в побудові програми, що буде подавляти шуми на зображенні.

1.4 Аналіз існуючих методів вирівнювання яскравості і контрастності зображень

Яскравість – світлова характеристика тіл. Відношення сили світла, що випромінюється поверхнею, до площі її проекції на площині перпендикулярної осі спостереження.

Контрастність – помітність предмета спостереження від навколишнього фону (монохроматичне випромінювання). Колірна контрастність – різновид оптичної контрастності, пов'язана з різницею відтінків кольорів.

Корекція яскравості та контрастності зображень. Зображення найчастіше є малоконтрастними. Слабкий контраст, як правило, обумовлений широким діапазоном відтворюваних яскравостей, що нерідко поєднується з нелінійністю характеристики передачі рівнів. Характер залежності зміни яскравості палітри пікселів від мінімального значення до максимального також впливає якість зображення. Оптимальною є лінійна функція зміни інтенсивності пікселів. При увігнутій характеристиці зображення буде темнішим, при опуклій світлішим. І в тому, і в іншому випадку ознаки об'єктів можуть бути спотворені та недостатньо добре ідентифіковані. Корекція (лінеаризація) яскравості палітри суттєво покращує якість зображення.

Мала контрастність може бути обумовлена й тим, що варіації функції яскравості пікселів на зображенні набагато менші за допустимий діапазон шкали яскравостей. У цьому випадку контрастність зображення підвищується шляхом розтягування реального динамічного діапазону яскравостей на всю шкалу за допомогою лінійного поелементного перетворення.

Деякі завдання обробки зображення пов'язані з перетворенням напівтонового зображення (багато градацій яскравості) на бінарне (дві градації). Перетворення здійснюється у тому, щоб скоротити інформаційну надмірність зображення, залишити у ньому лише інформацію, яка потрібна на вирішення конкретної задачі. У бінарному зображенні мають бути збережені певні деталі (наприклад, контури зображених об'єктів) та виключені несуттєві особливості (фон). Порогова обробка напівтонового зображення полягає у поділі всіх елементів зображення на два класи A_1 іA_2 за ознакою яскравості із кордоном A_{gr}, та у виконанні відповідної порогової фільтрації із заміною пікселів зображення на встановлену яскравість класів. Вибір кордону визначається видом гістограми яскравості вихідного зображення. Для найпростіших зображень типу креслень, машинописного тексту тощо, які мають бімодальний розподіл, межа встановлюється мінімум між модами розподілу. У загальному випадку зображення може бути багатомодальним, і якщо встановлюється досить надійна

відповідність між об'єктами та відповідними модами їхньої яскравості, то порогова фільтрація також може передбачати кілька класів яскравості пікселів.

Діапазон яскравості зображення в комп'ютері може відрізнятися від діапазону яскравостей вихідного, наприклад, через недостатню експозицію. Існує два можливі способи корекції яскравості. Згідно з першим способом зображення лінійно відображається в діапазоні яскравостей вихідного. Другий спосіб передбачає обмеження яскравості пікселів в обробленому зображенні максимальним і мінімальним граничними рівнями, і має ширше застосування. Присутність у зображенні найсвітліших і найтемніших тонів створює враження хорошої контрастності, проте зайва контрастність призводить до того, що максимальні градації впливають на середні тони, а більшість деталей зображення пофарбовані саме в середніх тонах і зайва контрастність може призвести до втрати цих деталей або утруднити їх виділення.

Гістограми яскравості. Інструментом для оцінки рівнів інтенсивності пікселів є гістограма – графічне відображення кількісної характеристики ймовірнісного розподілу інтенсивності (яскравості) пікселів у виділеній ділянці зображення. Максимальному значенню інтенсивності пікселів присвоюється рівень градації інтенсивності 255 (білий колір), темному - значення 0 (чорний колір). Інтенсивності в діапазоні від 0 до 255 мають лінійну шкалу зміни, або встановлюється відповідно до прийнятої функції зміни, наприклад, що посилює слабкі сигнали (градації сірого) і послаблює сильні сигнали (в області білого кольору), чим підвищується просторове і контрастне роздільна здатність зображення зони інтересу.

Відомий метод покращення зображень, заснований на обчисленні логарифму спектральних коефіцієнтів перетворення Фур'є вихідного зображення (обчислення кепстру). При зворотному перетворенні кепстру зображення відбувається вирівнювання гістограми зображення за рахунок логарифмічного перетворення спектра зображення.

Багато зображень характеризуються гістограмами з високою концентрацією ліній у певних зонах розподілу інтенсивності. Часто гістограма

розподілу яскравостей зображення має перекіс у бік малих рівнів (яскравість більшості елементів нижче за середню). Одним із методів поліпшення якості таких зображень є видозміна їхньої гістограми. Вирівнювання гістограми може бути здійснено на основі зведення в ступінь модуля спектральних коефіцієнтів Фур'є перетворення зображення, при цьому знак і фаза коефіцієнтів зберігається. Якщо позначити показник ступеня α , то при $\alpha < 1$ операція вилучення кореня ступеня α зменшує великі спектральні коефіцієнти та збільшує малі. Такий перерозподіл енергії у частотній площині зображення призводить до більш ефективного використання динамічного діапазону інтенсивностей пікселів зображення просторової області.

Вибір хорошої маски регулювання гістограми інтенсивності пікселів підвищує контраст, тим самим покращуючи контрастну роздільну здатність деталей. У програмах обробки є команди, що дозволяють встановлювати кольори при кольоровому картюванні зображень, що мають плавні або, навпаки, різкі переходи деталей, що відображаються в зоні інтересу. У поєднанні зі зверненням контрасту, що перетворює негативне зображення на позитивне, даний спосіб дозволяє також підвищити контраст дрібних і середніх деталей зображення.

Існує досить великий арсенал математичних моделей та алгоритмів, програмна реалізація яких дозволяє значно підвищити контрастну роздільну здатність зображень. Ці алгоритми засновані на процесах лінійної та нелінійної фільтрації зображень, що перетворює гістограму інтенсивності.

Деякі програми містять автоматичне встановлення оптимального співвідношення контрастності та яскравості, що підтримують необхідну чіткість зображення. При обробці зображень важливу роль відіграють алгоритми рангової фільтрації, що дають змогу усувати «розмитість» деталей (поліпшити їх фокусування) за рахунок вибору двовимірної $n \times n$ маски пікселів, виконання операції ранжування значень інтенсивності пікселів в межах маски, що задається, і присвоєння центральному пікселю значення, що дорівнює максимальному значенню по ранжиру. До виду рангової відноситься також

процедура медіанної фільтрації, що усуває у зображенні некорельовані випадкові сигнали та імпульсні перешкоди без "розмиття" різких перепадів яскравості на межах об'єктів.

Інший спосіб корекції яскравості палітри пов'язаний із інверсією вхідного зображення. Оскільки розрізняти слабкі сигнали на темному тлі досить складно, інверсна форма представлення таких зображень має іншу гістограму яскравостей, більш прийнятну для спостереження і візуальної ідентифікації. Такий спосіб називається інверсна медіанна фільтрація, що буде розглянута далі.

1.5 Аналіз яскравісного вирівнювання кадрів інверсним медіанним фільтром

Відомо, що медіанний фільтр при певному підборі розміру вікна може виконувати функції низькочастотного фільтра [7, 12]. При цьому результати високочастотної фільтрації, що відкидають крупноструктурні складові фона кадру, можуть бути отримані простим відніманням результату медіанної фільтрації кадру з вихідного кадру.

Іншими словами. Різниця у розмірах зображень цікавих об'єктів та фону (крупноструктурні складові зображень) забезпечує можливість використання медіанної фільтрації зображення для виділення крупноструктурних складових. Для вирівнювання фону зображення крупноструктурна складова, що отримана після медіанної фільтрації, віднімається з вихідного зображення. У зв'язку з цим даний метод був названий методом інверсної медіанної фільтрації.

Медіанна фільтрація – метод нелінійної обробки зображень [5 (стор. 234), 6 (стор. 50), 11 (стор. 194), 13 (стор. 129), 20 (стор. 118), 21 (стор. 224)] . Медіанний фільтр реалізується як процедура локальної обробки ковзним вікном заданих розмірів d_x та d_y, яке включає непарну кількість пікселів вихідного зображення A_{in} [5 (стор. 234), 6 (стор. 51), 21 (стор. 224), 14 (стор.120)]. У роботі

використовується медіанний фільтр із квадратним вікном розміром $d \times d$. Для кожного положення вікна, яскравості його пікселів упорядковуються за зростанням своїх значень [5 (стор. 234), 11 (стор. 195), 21 (стор. 224)]. Середній елемент у цій впорядкованій множині (медіана) замінює значення яскравості центрального пікселя у вікні для відфільтрованого зображення A_{med} [5 (стор. 234), 6 (стор. 51), 11 (стор. 195), 21 (стор. 224)] (рис. 2). Для медіани значень яскравості аналізованої групи з d^2 пікселів існує $(d^2-1)/2$ пікселів, менших або рівних йому за величиною і стільки ж більших або рівних [5 (стор. 234), 6 (стор. 51), 11 (стор. 195), 21 (стор. 224)].

Медіанний фільтр пригнічує зображення об'єктів на вихідному зображенні A_{in} у випадку, якщо площа зображення об'єктів становить менше половини площі d^2 апертури фільтра [5 (стор. 235), 6 (стор. 51), 11 (стор. 195), 21 (стор. 224)]. На Рис. 3а наведено приклад зображення з трьома гауссіанами, що представляють значення яскравості відповідних пікселів. Значення гаусіан $G_k(m,n)$ визначається відповідно до:

$$G_k(m,n) = a_k \exp\left\{-\frac{(m-m_k)^2+(n-n_k)^2}{2\sigma_k^2}\right\}$$

де k – номер гауссіани;

$a_k = 60$ – амплітуда k-ой гауссіани;

m_k, n_k – координати центра k-ой гауссіаны;

σ_k – параметр форми гауссіаны.

Гауссіани з параметром форми $\sigma_1 = 1$, $\sigma_2 = 2$ та $\sigma_3 = 3$ мають 28, 113 та 254 пікселів, у яких значення яскравості більше $0{,}01a_k$. На Рис. 3б; 3в і 3г (стор 49 даного файлу) представлені результати фільтрації зображення (рис. 3а) медіанними фільтрами з розмірами вікон 9х9, 17х17 та 27х27 пікселів. На відфільтрованих зображеннях видно як медіанний фільтр пригнічує зображення

об'єктів залежно від розміру вікна медіанного фільтра та площі зображення об'єктів.

Різниця у розмірах зображень цікавих об'єктів та фону (крупноструктурні складові зображень) забезпечує можливість використання медіанної фільтрації зображення для виділення крупноструктурних складових. Для вирівнювання фону зображення крупноструктурна складова, що отримана після медіанної фільтрації, віднімається з вихідного зображення.

Яскравісне вирівнювання зображення фону. Значення яскравостей пікселів зображення з вирівняним яскравістю фоном $A_{out}(m,n)$, без крупноструктурної складової, буде знайдено як різницю значень яскравостей пікселів вихідного зображення $A_{in}(m,n)$ та значень яскравостей пікселів відфільтрованого зображення $A_{med}(m,n)$:

$$A_{out}(m,n) = A_{in}(m,n) - A_{med}(m,n), \tag{1}$$

де $m = \overline{0, N_{CDDx} - 1}$ и $n = \overline{0, N_{CDDy} - 1}$.

Отримане, згідно (1), зображення з вирівняним за яскравістю фоном A_{out} матиме пікселі з негативними значеннями яскравості. Для усунення цього ефекту від значень яскравості всіх пікселів вирівняного зображення $A_{out}(m,n)$ віднімається значення пікселя з мінімальною яскравістю $A_{\min}$:

$$A_{out}(m,n) = A_{out}(m,n) - A_{\min}, \tag{2}$$

Через простоту, лаконічність та дієвість цього метода, саме він був у цій роботі взятий за основу алгоритму вирівнювання яскравості зображень.

1.6 Аналіз розробки програмного забезпечення

Прикладне програмне забезпечення (коротко додатки) — це комп’ютерне програмне забезпечення, призначене для виконання конкретних завдань, крім тих, що стосуються роботи самого комп’ютера[1], зазвичай для використання кінцевими користувачами. Прикладами таких програм є текстові процесори та медіаплеєри. Збірний іменник додаток доступний для всіх додатків у колективі. [2] Іншою основною класифікацією програмного забезпечення є системне програмне забезпечення та утиліти, пов'язані з комп'ютером («утиліти»).

Програми можуть бути в комплекті з вашим комп’ютером та його системним програмним забезпеченням, або вони можуть випускатися окремо і можуть бути зашифровані як власні, з відкритим вихідним кодом або на основі проекту.[3] Додатки, створені для мобільних платформ, називаються мобільними додатками.

В інформаційних технологіях додаток (додаток), прикладне або прикладне програмне забезпечення — це комп’ютерна програма, призначена для допомоги людям виконувати певну діяльність. Залежно від діяльності, для якої вона була розроблена, програма може обробляти текст, числа, звук, графіку та комбінації цих елементів. Деякі програмні пакети зосереджені на одному завданні, наприклад, обробці текстів; інші, які називаються інтегрованим програмним забезпеченням, включають кілька програм.

Програмне забезпечення, написане користувачем, адаптує систему до конкретних потреб користувача. Програмне забезпечення, написане користувачами, включає шаблони електронних таблиць, макроси текстового процесора, наукове моделювання, аудіо, графіку та сценарії анімації. Навіть фільтри електронної пошти є програмним забезпеченням для користувачів. Користувачі самі створюють це програмне забезпечення і зазвичай не помічають його важливості.

Розробка програмного забезпечення — це процес розробки, специфікації, проектування, програмування, документування, тестування та виправлення

помилок, пов’язаний зі створенням і підтримкою додатків, фреймворків або інших компонентів програмного забезпечення. Розробка програмного забезпечення передбачає написання та підтримку вихідного коду, але в ширшому сенсі вона включає всі процеси від концепції бажаного програмного забезпечення до остаточного прояву програмного забезпечення, як правило, у спланованому та структурованому процесі.[1] Розробка програмного забезпечення також включає дослідження, нові розробки, створення прототипів, модифікацію, повторне використання, переробку, технічне обслуговування або будь-яку іншу діяльність, яка призводить до створення програмних продуктів.[2]

Методики. Одна методологія розробки системи не обов’язково підходить для використання в усіх проектах.

Кожна з доступних методологій найкраще підходить для конкретних видів проектів, виходячи з різних технічних, організаційних, проектних і командних міркувань.[3]

Джерел ідей для програмних продуктів досить багато. Ці ідеї можуть виникнути в результаті дослідження ринку, включаючи демографічні дані потенційних нових клієнтів, існуючих клієнтів, потенційних клієнтів, які відмовилися від продукту, іншого внутрішнього персоналу з розробки програмного забезпечення або креативної третьої сторони. Ідеї для програмних продуктів зазвичай спочатку оцінюються спеціалістами з маркетингу на предмет економічної доцільності, відповідності існуючим каналам розповсюдження, можливого впливу на існуючі лінії продуктів, необхідних функцій і відповідності маркетинговим цілям компанії. На етапі маркетингової оцінки оцінюються припущення щодо витрат і часу. На початку першої фази приймається рішення про те, чи слід продовжувати проект на основі більш детальної інформації, отриманої спеціалістами з маркетингу та розробки.[4]

Важливим завданням при створенні ПЗ є аналіз вимог. Клієнти зазвичай мають абстрактне уявлення про те, що вони хочуть як кінцевий результат, але не знають, що програмне забезпечення повинно робити. Кваліфіковані та досвідчені інженери програмного забезпечення визнають неповні, неоднозначні або навіть

суперечливі вимоги на цьому етапі. Часта демонстрація коду в реальному часі може допомогти зменшити ризик того, що вимоги будуть неправильними.

«Хоча на етапі розробки вимог докладається багато зусиль, щоб гарантувати, що вимоги є повними та узгодженими, рідко це трапляється; залишаючи етап проектування програмного забезпечення найвпливовішим, коли йдеться про мінімізацію впливу нових або змінених вимог. Нестабільність вимог є складним, оскільки вони впливають на майбутні або вже реалізовані зусилля».[6]

Модель представлення — це структура, яка забезпечує точки зору на систему та її середовище, які будуть використовуватися в процесі розробки програмного забезпечення. Це графічне представлення основної семантики перегляду.

Мета точок зору та поглядів полягає в тому, щоб дозволити інженерам-людинам зрозуміти дуже складні системи та організувати елементи проблеми навколо областей знань. У розробці фізично інтенсивних систем точки зору часто відповідають можливостям і обов'язкам в інженерній організації.[7]

Моделювання бізнес-процесів і даних. Графічне представлення поточного стану інформації є дуже ефективним засобом для представлення інформації як користувачам, так і розробникам системи.

– Бізнес-модель ілюструє функції, пов’язані з бізнес-процесом, що моделюється, і організації, які виконують ці функції. Зображуючи дії та інформаційні потоки, створюється основа для візуалізації, визначення, розуміння та підтвердження природи процесу.

– Модель даних надає деталі інформації, яка має зберігатися, і є основною для використання, коли кінцевим продуктом є генерація коду комп’ютерного програмного забезпечення для програми або підготовка функціональної специфікації, щоб допомогти прийняти рішення про створення або купівлю комп’ютерного програмного забезпечення. Дивіться малюнок праворуч для прикладу взаємодії між бізнес-процесом і моделями даних.[8]

Зазвичай модель створюється після проведення співбесіди, яка називається бізнес-аналізом. Інтерв’ю складається з того, що фасилітатор ставить низку запитань, призначених для отримання необхідної інформації, яка описує процес. Інтерв'юера називають фасилітатором, щоб підкреслити, що інформацію надають саме учасники. Фасилітатор повинен мати певні знання про цікавий процес, але це не так важливо, як мати структуровану методологію, за якою ставлять запитання експерту з процесу. Методологія важлива, оскільки зазвичай команда фасилітаторів збирає інформацію по всьому об’єкту, а результати інформації від усіх інтерв’юерів повинні збігатися разом після завершення.[8]

Моделі розробляються як визначення поточного стану процесу, у цьому випадку кінцевий продукт називається моделлю знімка «як є», або набір ідей щодо того, що повинен містити процес, в результаті чого «що може» -be" модель. Генерація моделей процесів і даних може бути використана для визначення того, чи існуючі процеси та інформаційні системи надійні та потребують лише незначних модифікацій чи вдосконалень, чи потрібна реінжиніринг як коригувальна дія. Створення бізнес-моделей — це більше, ніж спосіб перегляду чи автоматизації вашого інформаційного процесу. Аналіз можна використовувати, щоб кардинально змінити те, як ваш бізнес чи організація веде свою діяльність.[8]

Автоматизоване програмне забезпечення. Комп’ютеризована інженерія програмного забезпечення (CASE) у галузі програмної інженерії — це наукове застосування набору програмних інструментів і методів для розробки програмного забезпечення, що призводить до отримання високоякісних, бездефектних і зручних програмних продуктів.[9]] Це також відноситься до методів розробки інформаційних систем разом з автоматизованими інструментами, які можна використовувати в процесі розробки програмного забезпечення.[10] Термін «комп’ютерна розробка програмного забезпечення» (CASE) може стосуватися програмного забезпечення, яке використовується для автоматизованої розробки системного програмного забезпечення, тобто

комп’ютерного коду. Функції CASE включають аналіз, проектування та програмування. Інструменти CASE автоматизують методи проектування, документування та створення структурованого комп’ютерного коду потрібною мовою програмування.[11]

Дві ключові ідеї автоматизованої розробки програмного забезпечення (CASE): [12]

– Сприяти комп’ютерній допомозі в процесах розробки та обслуговування програмного забезпечення, а також

– Інженерний підхід до розробки та супроводу програмного забезпечення.

Існують типові інструменти CASE для керування конфігураціями, моделювання даних, перетворення моделі, рефакторингу, створення вихідного коду.

Мова моделювання — це будь-яка штучна мова, яку можна використовувати для вираження інформації, знань або систем у структурі, яка визначається послідовним набором правил. Правила використовуються для тлумачення значення компонентів у структурі. Мова моделювання може бути графічним або текстовим.[13]

Парадигма програмування — це фундаментальний стиль комп’ютерного програмування, який, як правило, не диктується методологією управління проектами (такими як водоспад або гнучкість). Парадигми відрізняються концепціями та абстракціями, які використовуються для представлення елементів програми (таких як об’єкти, функції, змінні, обмеження) та кроків, які складають обчислення (таких як призначення, оцінка, продовження, потоки даних). Іноді концепції, стверджувані парадигмою, спільно використовуються в проектуванні архітектури системи високого рівня; в інших випадках область застосування парадигми програмування обмежена внутрішньою структурою конкретної програми або модуля. Приклад: об’єктно-орієнтований дизайн (OOD) Грейді Буча, також відомий як об’єктно-орієнтований аналіз і дизайн (OOAD). Модель Booch включає шість діаграм: клас, об'єкт, перехід стану, взаємодія, модуль і процес.[14]

2 ВИБІР ІНСТРУМЕНТАРІЮ РОЗРОБКИ

2.1 Огляд мови програмування

Python — це інтерпретована мова програмування загального рівня високого рівня. Філософія дизайну Python підкреслює читабельність коду за допомогою значного відступу. Його мовна структура та об’єктно-орієнтований підхід розроблені, щоб допомогти програмістам писати чіткі логічні коди для малих і великих проектів.

Python динамічно збирає та збирає сміття. Він підтримує декілька парадигм програмування, включаючи структурне (включаючи процедурне), об’єктно-орієнтоване та функціональне програмування. Через повну стандартну бібліотеку Python часто описують як мову, що включає батареї.

Як наступник мови програмування ABC, Гвідо ван Россум почав вивчати Python наприкінці 1980-х і вперше був випущений в 1991 році під назвою Python 0.9.0. [32] Python 2.0 був випущений у 2000 році і представив нові функції, такі як розуміння списку та систему збору сміття з використанням підрахунку посилань, Його буде припинено у версії 2.7.18 у 2020 році [33]. Python 3.0 був випущений у 2008 році. Це основна версія мови, яка не повністю сумісна з протилежною, і більшість коду Python 2 не може залишатися незмінною на Python 3.

Python завжди був однією з найпопулярніших мов програмування.

Python був задуманий наприкінці 1980-х років Гвідо ван Россумом з Centrum Wiskunde & Informatica (CWI) в Нідерландах, як наступник мови програмування ABC, натхненної SETC, здатної обробляти винятки та взаємодіяти з операційною системою Amoeba. Його реалізація почалася в грудні 1989 року. Ван Россум був повністю відповідальним за проект як провідний розробник до 12 липня 2018 року, коли він оголосив про свою «постійну відпустку» і отримав титул «Диктатора, дружнього до життя» Python. Він відображає свою довгу історію через спільноту Python. Довгострокове

зобов’язання працювати в якості головного менеджера проекту. Тепер він поділяє своє лідерство як член наглядової ради з п’яти осіб. У січні 2019 року активні розробники ядра Python обрали Бретта Кеннона, Ніка Коглана, Баррі Варшава, Керол Віллінг і Ван Россума до ради директорів із п’яти осіб. Відтоді Гвідо ван Россум зняв свою кандидатуру в раду директорів 2020 року.

Python 2.0 був випущений 16 жовтня 2000 року з багатьма важливими новими функціями, включаючи збірник сміття для виявлення циклу та підтримкою Unicode.

Python 3.0 був випущений 3 грудня 2008 року. Це серйозна редакція мови і не повністю сумісна з іншою стороною. Багато з його основних функцій було перенесено до Python версій 2.6.x і 2.7.x. Реліз Python 3 містить утиліту 2to3, яка автоматично (принаймні частково) перетворює код Python 2 на Python 3.

Термін дії Python 2.7 спочатку був встановлений у 2015 році, а потім перенесений на 2020 рік через побоювання, що велика кількість існуючого коду може бути нелегкою в перенесені на Python 3. Він не випускатиме більше виправлень безпеки та інших покращень. Наприкінці життєвого циклу підтримується лише Python 3.6.x і новіших версій.

Python 3.9.2 і 3.8.8 прискорені, оскільки всі версії Python мають проблеми з безпекою, що призводить до можливого віддаленого виконання коду та отруєння веб-кешу.

Python — це мова програмування з кількома парадигмами. Він повністю підтримує об’єктно-орієнтоване програмування та структурне програмування, а багато його функцій підтримують функціональне програмування та аспектно-орієнтоване програмування (включаючи метапрограмування та метаоб’єктний (магічний метод)). Розширення підтримує багато інших парадигм, включаючи дизайн контрактів і логічне програмування.

Python використовує комбінацію динамічного введення тексту та підрахунку посилань і збирач сміття, який визначає цикли для керування пам’яттю. Він також має динамічне розділення імен (пізнє прив’язування) для зв’язування імен методів і змінних під час виконання програми.

Python розроблено для забезпечення певної підтримки функціонального програмування в традиції Lisp. Він має функції фільтрації, відображення та скорочення; перераховує розуміння, словник, набір і вираз генератора. Стандартна бібліотека має два модулі (itertools і functools), які реалізують функціональні інструменти, запозичені з Haskell і Standard ML.

Дзен Python (PEP 20) підсумовує основні концепції мови, включаючи такі сентенції:

– красиве - краще, ніж потворне;

– явне краще, ніж неявне;

– просте - краще, ніж складне;

– складний - це краще, ніж складний;

– підрахунок читабельності.

Python не має всіх своїх функцій, вбудованих у його ядро, але розроблено, щоб бути дуже розширюваним (з модулями). Ця компактна модульність робить його особливо популярним як спосіб додавання програмованих інтерфейсів до існуючих програм. Ідея Ван Россума про невелику хост-мову, велику стандартну бібліотеку та легко розширюваний перекладач походить від його незадоволення ABC, який підтримує протилежний підхід. Python прагне до простішого та стислого синтаксису та граматики, надаючи розробникам можливість вибору методів кодування. На відміну від девізу Perl «Є більше одного способу зробити це», Python висвітлює філософію дизайну: «Повинен бути один – бажано лише один – очевидний спосіб зробити це». [69] Алекс Мартеллі, член Python Software Foundation і автор книги про Python, написав: «У культурі Python опис чогось як «розумного» не вважається компліментом».

Розробники Python прагнуть уникнути передчасної оптимізації та відмовляються виправляти некритичні частини референсної реалізації CPython, які забезпечують невелике збільшення швидкості за ціною наочності. Коли швидкість важлива, програмісти Python можуть перемістити важливі за часом функції в розширення, написані на таких мовах, як C, або використовувати PyPy,

компілятор «точно вчасно». Також доступний Cython, який перетворює скрипти Python на C і здійснює прямі виклики API рівня C до інтерпретатора Python.

Важлива мета для розробників Python — зробити його веселим. Це знайшло відображення в назві мови — данині британській комедійній групі «Монті Пайтон» — і іноді грайливому підході до підручників та довідкових матеріалів, таких спаму та яєць (із відомого етюду Монті Пайтона) стандартних foo and bar.

Поширеним новим словом у спільноті Python є python, яке може мати широкий спектр значень, пов’язаних зі стилем програмування. Сказати, що код є pythonic, означає, що він добре використовує ідіоми Python, що є природним, або що він демонструє вільну мову, яка відповідає мінімалістичній філософії Python і підкреслює читабельність. На відміну від цього, код, який важко зрозуміти або прочитати як приблизну копію іншої мови програмування, називається не Python.

Користувачів і шанувальників Python, особливо тих, кого вважають обізнаними або досвідченими, часто називають Pythonists. Python має бути легкою для читання мовою. Його формат візуально не захаращений, часто використовуються ключові слова англійською мовою, тоді як інші мови використовують розділові знаки. На відміну від багатьох інших мов, він не використовує дужки для розділення блоків і допускає крапку з комою після операторів, але рідко використовується, якщо взагалі використовується. У порівнянні з C або Pascal, він має менше граматичних винятків і особливих випадків.

Python використовує пробіли замість фігурних дужок або ключових слів для розділення блоків. Збільшення відступу відбувається після певних операторів; зменшення відступу означає кінець поточного блоку. Тому візуальна структура програми точно відображає семантичну структуру програми. Цю функцію іноді називають «зовнішніми» правилами, і в деяких інших мовах вона є, але в більшості мов відступ не має семантичного значення. Рекомендований розмір відступу – чотири пробіли.

Оператори Python включають (серед іншого):

– Використовуйте знак рівності = оператор присвоєння.

– оператор if, умовно виконати блок коду, а також else та elif (скорочення від else-if).

– Оператор for для перегляду ітераційного об’єкта шляхом захоплення кожного елемента в локальну змінну для використання вкладеним блоком.

– Поки умова істинна, виконується оператор while блоку коду.

– Оператор try, який дозволяє витягам, що містяться у вкладених блоках коду, захоплювати й обробляти вміст, відмінний від речень; він також гарантує, що незалежно від того, як виходить блок, чистий код у блоці в кінцевому підсумку буде виконано.

– Оператор raise використовується для отримання вказаного винятку або повторного виклику перехопленого винятку.

– Оператор класу, який виконує блок коду та приєднує свій локальний простір імен до класу для використання в об’єктно-орієнтованому програмуванні.

– Оператор def, який визначає функцію або метод.

– Оператор Python 2.5, випущений у вересні 2006 р. [82], охоплює блок коду в диспетчері контексту (наприклад, отримання блокування перед запуском блоку коду і зняття блокування після цього або відкриття файлу, а потім його закриття), що дозволяє Поведінка, подібна до пошуку ресурсів, — це ініціалізація (RAII), яка замінює поширену ідіому try / finally.

– Оператор break виходить з циклу.

– Оператор continue пропускає цю ітерацію і переходить до наступного елемента.

– Оператор del видаляє змінну, а це означає, що посилання з імені на значення видаляється, а спроби використати змінну призведуть до помилки. Видалену змінну можна повторно призначити.

– Оператор проходу, виконує функцію NOP. Це синтаксично необхідно для створення порожнього блоку коду.

– Оператори тверджень, які використовуються для перевірки умов, що застосовуються під час налагодження.

– Оператор yield, який повертає значення з функції генератора. У Python 2.5 yield також є оператором. Ця форма використовується для реалізації спільного плану.

– Оператор повернення, який використовується для повернення значення з функції.

– Оператор імпорту, що використовується для імпорту модуля, чия функція або змінна може використовуватися в поточній програмі. Існує три способи імпорту: імпорт <ім’я модуля> [як <псевдонім>] або за допомогою <ім’я модуля> імпорт * або за допомогою <ім’я модуля> імпорт <визначення 1> [як <псевдонім 1>], <визначення 2> [як <псевдонім 2>], ...

– Оператор присвоєння (=) діє шляхом прив'язки імені як посилання на окремий динамічно розподілений об'єкт. Потім змінні можуть бути відскочені в будь-який час до будь-якого об’єкта. У Python ім'я змінної є загальним власником посилання і не має фіксованого типу даних, пов’язаного з ним. Однак у даний момент часу змінна буде посилатися на якийсь об'єкт, який матиме тип. Це називається динамічним набором тексту і протиставляється статично набраним мовам програмування, де кожна змінна може містити лише значення певного типу.

Python не підтримує оптимізацію хвостових викликів або першокласні розширення, і, за словами Гвідо ван Россума, ніколи не буде. Однак, розширивши генератор Python, у версії 2.5 надається краща підтримка функцій, подібних до корутина. До 2,5 генераторів є ледачими ітераторами, інформація передається від генераторів в одному напрямку. З Python 2.5 ви можете передавати інформацію назад до функції генератора, а з Python 3.3 ви можете передавати інформацію через кілька рівнів стека.

Деякі вирази Python подібні до виразів у таких мовах, як C і Java, а деякі ні:

– Додавання, віднімання та множення однакові, але поведінка ділення

різна. У Python є два типи розділів. Це поділ на поверхню (або ціле поділ) // і з плаваючою комою/діленням. Python також використовує оператори ** для просування.

– Новий оператор @ infix був представлений у Python 3.5. Він призначений для використання такими бібліотеками, як NumPy для множення матриць.

– У Python 3.8 введено синтаксис: =, який називається «оператором моржа». Він призначає значення змінним як частину більшого виразу. [91]

– У Python == порівнює з Java за значенням, Java порівнює значення за значенням і порівнює об'єкти за посиланням. (У Java ви можете використовувати метод equals для порівняння значень об’єктів ().) Оператор Python is можна використовувати для порівняння ідентифікаторів об’єктів (порівняння посилань). У Python порівняння можна ланцюжком, наприклад a <= b <= c.

– Python використовує слова та, або, не для своїх булевих операторів, а не для символічного &&, ||,! використовується в Java та C.

– Python має тип виразу, що називається розумінням списку, а також більш загальний вираз, який називається генераторським виразом.

– Анонімні функції реалізовані за допомогою лямбда-виразів; однак вони обмежені тим, що тіло може бути лише одним виразом.

– Умовні вирази в Python пишуться як x, якщо c else y (відрізняється за порядком операндів від оператора c? X: y, загального для багатьох інших мов).

– Python розрізняє списки та кортежи. Список записується як [1, 2, 3], змінюється і не може використовуватися як ключі словника (ключі словника повинні бути незмінними в Python). Кортеж записується як (1, 2, 3) і є незмінним, тому, поки всі елементи кортежу незмінні, його можна використовувати як ключ словника. Оператор + можна використовувати для об’єднання двох кортежів, що безпосередньо не змінює їх вміст, а створює новий кортеж, що містить два надані елементи кортежу. Тому, враховуючи, що змінна t спочатку дорівнює (1, 2, 3), коли виконується t = t + (4, 5), спочатку обчислюється t + (4, 5), щоб отримати (1, 2, 3 , 4, 5), а потім призначити його назад до t, тим самим фактично «змінюючи вміст t», відповідаючи інваріантній природі об’єкта кортежу. Дайте зрозуміти,

що кортежі в контексті не потребують дужках.

– Python має послідовність розпакування, в якій кілька виразів, кожен з яких обчислює все, що можна призначити (змінні, атрибути запису тощо), пов’язуються в тій самій формі, що й літерал кортежу, і в цілому поміщають твердження твердження зліва Сторона середнього розміру. Оператор очікує повторюваний об’єкт праворуч від знака рівності, який виводить таку ж кількість значень, як і вираз, наданий для запису під час сортування, і фільтрує його, і призначає кожне створене значення відповідному виразу зліва. .

– У Python є оператор % «формат рядка». Це схоже на рядок printf в C, наприклад "спам =% s egg =% d"% ("blah", 2) оцінюється як "спам = blah-eggs = 2". У Python 3 і 2.6+ це доповнюється методом format() класу str, наприклад "спам = {0}яйця = {1}". Формат («Зачекайте», 2). Python 3.6 додав "f-рядки": blah = "blah"; egg = 2; f'spam = {blah} egg = {eggs}'.

– Рядки в Python можна об’єднати шляхом «додавання» (той самий оператор, що й додавання значень цілого чи плаваючої коми). Наприклад, "спам" + "яйце" повертає "спам". Навіть якщо ваш рядок містить числа, вони все одно додаються як рядки, а не цілі числа. Наприклад, "2" + "2" повертає "22".

– Python має різні типи рядкових літералів:

– Рядки розділені одинарними або подвійними лапками. На відміну від оболонки Unix, мови Perl і Perl, які впливають на Perl, одинарні та подвійні лапки працюють однаково. Обидва типи рядків використовують зворотну косу риску (\) як вихідні символи. Інтерполяція рядка вже доступна як "відформатований рядковий літерал" у Python 3.6.

– Рядки з подвійними лапками, які починаються та закінчуються серією з трьох одинарних або подвійних лапок. Вони можуть охоплювати кілька рядків і функціонувати, як тут документи в оболонках, Perl і Ruby.

– Сирі різновиди рядків, що позначаються префіксом рядкового літералу перед r. Послідовності втечі не інтерпретуються; отже, необроблені рядки корисні там, де буквенні зворотні слеші є загальними, такі як регулярні вирази та шляхи у стилі Windows. Порівняйте "@ -citation" у C #.

– Python має індекси масивів та вирази нарізування масивів у списках, що позначаються як [ключ], [старт: зупинка] або [старт: зупинка: крок]. Індекси базуються на нулі, а негативні індекси відносно кінця. Зрізи беруть елементи від початкового індексу до, але не включаючи, індексу зупинки. Третій параметр зрізу, який називається кроком або кроком, дозволяє пропускати та обертати елементи. Індекси зрізів можуть бути опущені, наприклад, [:] повертає копію всього списку. Кожен елемент зрізу є неглибокою копією.

Python чітко розрізняє вирази та вирази, що відрізняється від мов, таких як Common Lisp, Scheme або Ruby. Це може призвести до дублювання певних функцій.

Метод для об’єкта — це функція, приєднана до класу об’єкта; синтаксис instance.method (аргумент) — це синтаксичний цукор Class.method (екземпляр, аргумент) для поширених методів і функцій. Методи Python мають явний параметр self для доступу до даних екземпляра, що є протилежністю неявного self (або цього) в деяких інших об’єктно-орієнтованих мовах програмування (наприклад, C++, Java, Objective-C або Ruby).

Python використовує набір качок і має типізовані об’єкти, але нетиповані імена змінних. Обмеження типу не перевіряються під час компіляції; навпаки, операції над об’єктом можуть бути невдалими, а це означає, що об’єкт не має відповідного типу. Хоча Python динамічно типізується, він забороняє невизначені операції (наприклад, додавання чисел до рядка) і не намагається зрозуміти їх тихо.

Python дозволяє програмістам визначати власні типи, використовуючи класи, які найчастіше використовуються в об’єктно-орієнтованому програмуванні. Новий екземпляр класу створюється шляхом виклику класу (наприклад, SpamClass() або EggsClass()).Клас — це екземпляр типу метакласу (сам екземпляр), який дозволяє виконувати метапрограмування та відображати.

До версії 3.0 Python мав два типи класів: старі та нові. Синтаксис двох стилів однаковий, різниця полягає в тому, чи успадковується об’єкт класу безпосередньо чи опосередковано (усі класи нового стилю успадковуються від

об’єкта і є екземплярами типу). У Python 2, починаючи з Python 2.2, ви можете використовувати ці два типи класів. Класи старого стилю були ліквідовані в Python 3.0.

Довгостроковий план передбачає підтримку інкрементного введення, а в Python 3.5 синтаксис мови дозволяє вказувати статичні типи, але вони не перевіряються в реалізації CPython за замовчуванням. Експериментальна додаткова статична перевірка типу муру підтримує перевірку типу під час компіляції.

CPython є еталонною реалізацією Python. Він написаний на C, відповідає стандарту C89 і має кілька вибраних функцій C99 (в більш пізніх версіях C він вважається застарілим; CPython включає власні розширення C, але сторонні розширення не обмежуються старими C версія, наприклад, може бути реалізована на C11 або C++). Він компілює програми Python у проміжні байт-коди, які потім виконуються його віртуальною машиною. CPython постачається з великою стандартною бібліотекою, яка написана сумішшю C і рідного Python. Він працює на багатьох платформах, включаючи Windows (починаючи з Python 3.9, інсталятор Python навмисно не зміг встановити в Windows 7 і 8; Windows XP підтримувалася до Python 3.5) і більшість сучасних Unix-подібних систем, включаючи macOS (і Apple M1). Mac), починаючи з Python 3.9.1, з експериментальним інсталятором) та неофіційною підтримкою, наприклад VMS. Переносність платформи є одним з її головних пріоритетів, і навіть OS/2 і Solaris підтримувалися під час Python 1 і 2; з тих пір рівень підтримки багатьох платформ знизився.

Python в основному розробляється за допомогою Програми покращення Python (PEP), яка є основним механізмом, який надає ключові нові можливості, покращує думку спільноти щодо проблем і фіксує рішення щодо дизайну Python. Стиль кодування Python наведено в PEP 8. Відмінні PEP розглядаються та коментуються спільнотою Python та радою директорів.

Покращення мови відповідають розробці довідкової реалізації CPython. Список розсилки python-dev є основним форумом для розробки мови. Конкретні

питання обговорюються в інструменті відстеження помилок Roundup на bugs.python.org. Спочатку розробка велася у власному сховищі вихідного коду Mercurial, поки Python не перейшов на GitHub у січні 2017 року.

Відповідно до того, наскільки збільшилася кількість версій, існує три типи загальнодоступних випусків CPython:

Зворотна мовна версія несумісна. У цьому випадку код буде зламаний і потрібне ручне перенесення. Перша частина номера версії збільшена. Ці версії рідкісні-версія 3.0 вийшла через 8 років після виходу версії 2.0.

Основна або «функціональна» версія виходить приблизно кожні 18 місяців, але з прийняттям щорічної каденції випуску, починаючи з Python 3.9, очікується, що випускатиме її щорічно. Вони в основному сумісні, але вводять нові функції.

Друга частина номера версії збільшена. Патч підтримує всі основні версії протягом кількох років після його випуску.

Версії, які не містять нових функцій, випускаються приблизно кожні 3 місяці і випускаються, коли було виправлено достатню кількість помилок з моменту останнього випуску. Уразливості безпеки також були усунені в цих версіях. Третя і остання частина номера версії збільшено.

Багато альфа-, бета- та реліз-кандидатів також попередньо переглядаються та тестуються перед остаточним релізом. Хоча кожна проблема має приблизний графік, якщо код не готовий, вони зазвичай затримуються. Команда розробників Python відстежує стан коду, виконуючи велику кількість модульних тестів під час процесу розробки.

Головна академічна конференція для Python — PyCon. Також існують спеціальні навчальні програми Python, наприклад Pyladies.

Pythons 3.10 припиняє wstr (буде видалено в Python 3.12; це означає, що розширення Python потрібно буде змінити до того часу), а також планує додати відповідність шаблону до мови.

З 2003 року Python є однією з десяти найпопулярніших мов програмування в індексі програмного забезпечення TIOBE, а станом на лютий 2021 року це третя

за популярністю мова (після Java та C). Вона була обрана мовою програмування року («Річний найвищий ріст рейтингу») у 2007, 2010, 2018 та 2020 роках (єдина мова, яка робила це чотири рази).

Емпіричні дослідження показали, що для проблем програмування, включаючи маніпулювання рядками та пошук у словнику, мови сценаріїв, такі як Python, є більш продуктивними, ніж звичайні мови, такі як C і Java, і виявилося, що споживання пам'яті часто краще, ніж у Java, і немає краще, ніж C або Наскільки поганий C++".

Великі організації, які використовують Python, включають Wikipedia, Google, Yahoo!, CERN, NASA, Facebook, Amazon, Instagram, Spotify, а також деякі менші організації, такі як ILM та ITA. Сайт соціальних новин Reddit в основному написаний на Python.

Python може бути мовою сценаріїв для веб-програм, наприклад mod_wsgi для веб-сервера Apache. Стандартні API були розроблені для полегшення роботи цих програм через інтерфейс шлюзу веб-сервера. Веб-фреймворки, такі як Django, Pylons, Pyramid, TurboGears, web2py, Tornado, Flask, Bottle і Zope, підтримують розробників для розробки та підтримки складних програм. Pyjs і IronPython можна використовувати для розробки клієнтської сторони програм на основі Ajax. SQLAlchemy можна використовувати як перетворювач даних у реляційних базах даних. Twisted — це фреймворк для програмного зв’язку між комп’ютерами, який використовується (наприклад) Dropbox.

Такі бібліотеки, як NumPy, SciPy і Matplotlib, дозволяють ефективно використовувати Python в наукових обчисленнях, тоді як спеціалізовані бібліотеки, такі як Biopython і Astropy, забезпечують функції, залежні від домену.

Програмне забезпечення SageMath-Math з інтерфейсом ноутбука, яке можна запрограмувати на Python: його бібліотека охоплює багато аспектів математики, включаючи алгебру, комбінаторику, числову математику, теорію чисел і числення. OpenCV має палітурні прив'язки з багатим набором функцій для комп'ютерного зору та обробки зображень.

Python часто використовується в проектах штучного інтелекту та машинного навчання, включаючи такі бібліотеки, як TensorFlow, Keras, Pytorch і Scikit-learn. Python, як мова сценаріїв з модульною архітектурою, простим синтаксисом і багатими інструментами обробки тексту, часто використовується для обробки природних мов.

Python був успішно інтегрований у багато програмних продуктів як мова сценаріїв, включаючи програмне забезпечення кінцевих елементів, таке як Abaqus, засоби моделювання 3D-параметрів, такі як FreeCAD, пакети 3D-анімації, такі як 3ds Max, Blender, Cinema 4D, Lightwave, Houdini, Maya, modo, MotionBuilder, Softimage, композитор візуальних ефектів Nuke, програми для 2D-зображень (такі як GIMP, Inkscape, Scribus і Paint Shop Pro) і програми для нот (наприклад, автор і група).

Налагоджувач GNU використовує Python як хороший принтер для відображення складних структур, таких як контейнери C++.

Esri рекламує Python як найкращий вибір для написання сценаріїв у ArcGIS. Він також використовувався в багатьох відеоіграх і був прийнятий як перша з трьох доступних мов програмування в Google App Engine, двома іншими є Java і Go.

Багато операційних систем використовують Python як стандартний компонент. Він постачається з більшістю дистрибутивів Linux, AmigaOS 4 (з використанням Python 2.7), FreeBSD (як пакет), NetBSD, OpenBSD (як пакет) і macOS, і може використовуватися з командного рядка (термінал). У багатьох дистрибутивах Linux використовуються інсталятори, написані на Python: Ubuntu використовує інсталятор Ubiquity, а Red Hat Linux і Fedora — інсталятор Anaconda. Gentoo Linux використовує Python у своїй системі керування пакетами Portage.

Python широко використовується для інформаційної безпеки, включаючи розробку експлойтів.

Більшість додатків Sugar Laptop XO, які зараз розробляє Sugar Labs, написані на Python.

Проект одноплатної комп’ютерної програми Raspberry Pi прийняв Python як основну мову програмування користувачів.

LibreOffice включає Python і має намір замінити Java на Python. Його постачальник сценаріїв Python є основною особливістю версії 4.0 від 7 лютого 2013 року.

Виходячи зі сфери застосування та величезних можливостей мови Python, описаної вище, можна зробити висновок, що вона універсальна. Ось чому ця мова програмування була обрана для виконання цієї роботи.

2.2 Огляд середивища розробки

PyCharm — це інтегроване середовище розробки (IDE) для комп’ютерного програмування, особливо для Python. Його розробила чеська компанія JetBrains. Він забезпечує аналіз коду, графічний налагоджувач, інтегрований модуль тестування та інтеграцію системи контролю версій (VCS), а також підтримує використання Django для веб-розробки та Anaconda для аналізу даних.

PyCharm є кросплатформним і має версії для Windows, macOS та Linux. Спільнота доступна за ліцензією Apache, а також є професійна версія з додатковими функціями, випущена за власною ліцензією.

Бета-версія була випущена в липні 2010 року, а версія 1.0 з'явилася через 3 місяці. Версія 2.0 була випущена 13 грудня 2011 року, версія 3.0 – 24 вересня 2013 року, версія 4.0 – 19 листопада 2014 року.

PyCharm Community Edition — версія PyCharm з відкритим кодом, яка була запущена 22 жовтня 2013 року.

2.3 Огляд додаткового інструментарію

Для подання моделі потоків даних використовується CASE-засіб ERWin. З його допомогою при проектуванні моделі інформаційної системи «Тимчасовий притулок для тварин».

OpenCV (Бібліотека комп’ютерного бачення з відкритим вихідним кодом) — це бібліотека функцій програмування, спрямованих головним чином на комп’ютерне зір у реальному часі.[1] Спочатку розроблений Intel, пізніше його підтримувала Willow Garage, а потім Itseez (який пізніше був придбаний Intel [2]). Бібліотека є кросплатформною та безкоштовною для використання за ліцензією Apache 2 з відкритим кодом. Починаючи з 2011 року OpenCV підтримує прискорення GPU для операцій у реальному часі.[3]

Історія. Офіційно запущений у 1999 році проект OpenCV спочатку був ініціативою Intel Research для вдосконалення додатків із інтенсивним використанням процесора, частиною серії проектів, включаючи трасування променів у реальному часі та 3D-дисплеї.[4]

Перша альфа-версія OpenCV була оприлюднена на Конференції IEEE з комп’ютерного зору та розпізнавання образів у 2000 році, а п’ять бета-версій було випущено між 2001 та 2005 роками. Перша версія 1.0 була випущена в 2006 році. Попередня версія 1.1 » вийшов у жовтні 2008 року.

Другий великий випуск OpenCV був випущений у жовтні 2009 року. OpenCV 2 містить значні зміни в інтерфейсі C++, спрямовані на простіші, більш безпечні шаблони, нові функції та кращі реалізації для існуючих з точки зору продуктивності (особливо на кількох платформах). основні системи).

Офіційні випуски тепер відбуваються кожні шість місяців [6], а розробкою зараз займається незалежна російська команда за підтримки комерційних корпорацій.

У серпні 2012 року підтримку OpenCV взяла на себе некомерційна організація OpenCV.org, яка підтримує сайт розробників[7] і користувачів.[8]

У травні 2016 року Intel підписала угоду про придбання Itseez [9], провідного розробника OpenCV [10].

У липні 2020 року OpenCV анонсувала та розпочала кампанію на Kickstarter для OpenCV AI Kit, серії апаратних модулів і доповнень до OpenCV, що підтримують Spatial AI.

Мова програмування. OpenCV написаний мовою C++, і його основний інтерфейс є мовою C++, але він все ще зберігає менш повний, але розширений старий інтерфейс C. Усі нові розробки та алгоритми з'являються в інтерфейсі C++. Є прив'язки в Python, Java та MATLAB/OCTAVE.

API для цих інтерфейсів можна знайти в онлайн-документації.[12] Було розроблено оболонки кількома мовами програмування, щоб заохотити впровадження ширшої аудиторії. У версії 3.4 прив'язки JavaScript для вибраної підмножини функцій OpenCV було випущено як OpenCV.js для використання на веб-платформах.[13]

Апаратне прискорення. Якщо бібліотека знаходить інтегровані примітиви продуктивності Intel у системі, вона використовуватиме ці власні оптимізовані процедури для прискорення.

Інтерфейс GPU на основі CUDA розробляється з вересня 2010 року.[14]

Інтерфейс GPU на основі OpenCL розробляється з жовтня 2012 року[15], документацію для версії 2.4.13.3 можна знайти на docs.opencv.org.[16]

Підтримка ОС. OpenCV працює на таких настільних операційних системах: Windows, Linux, macOS, FreeBSD, NetBSD, OpenBSD. OpenCV працює на таких мобільних операційних системах: Android, iOS, Maemo[17], BlackBerry 10.[18] Користувач може отримувати офіційні випуски від SourceForge або брати найновіші джерела від GitHub.[19] OpenCV використовує CMake.

3 ПРОЕКТУВАННЯ І РОЗРОБКА СИСТЕМИ

3.1 Побудова контексної діаграми IDEF0

IDEF0 — це методологія графічного опису систем і процесів діяльності організації як безліч взаємозв'язаних функцій. Вона дозволяє досліджувати функції організації, не зв'язуючи їх з об'єктами, що забезпечують їх реалізацію.

У стандарті IDEF0 за допомогою входу показують об'єкти — інформаційні та матеріальні потоки, які перетворюються в бізнес-процес. З допомогою управління показуються об'єкти — матеріальні та інформаційні потоки, які не преобразуются в процесі, потрібні для його виконання. За допомогою механізму IDEF0 можна відображати інструменти та ресурси, за допомогою яких реалізується бізнес-процес (наприклад, технічні засоби, люди, інформаційні системи тощо). Вихід бізнес-процесу, описаного в стандарті IDEF0, повністю відповідає змісту виходу процесу, описаному за допомогою DFD-схеми.

Діаграма IDEF0 A-0 представляє подання на рівні контексту входів, керування, виходів і механізмів (ICOM) для певної функції у вашій логічній моделі. Частина набору представлень поведінки (логічна архітектура), IDEF0 A-0 є контекстним представленням, яке доповнює повний IDEF0 та інші представлення поведінки на будь-якому рівні вашої логічної архітектури.

Діаграма IDEF0 доступна для елементів класу Function (а також будь-яких інших підкласів ProcessingUnit).

На діаграмі A-0 зазвичай можна побачити тунелювання. Тунелювання — це техніка IDEF0 для приховування ICOM у частині моделі. Використання круглих дужок навколо голови чи хвоста стрілки зображує тунель у IDEF0. Дужка навколо головки стрілки, яка входить у вікно функції, вказує на те, що ICOM, пов’язаний із цією стрілкою, не буде видно під час декомпозиції цієї функції. Якщо ICOM знову з’явиться, його хвіст матиме дужки. CORE вирішує цю проблему автоматично. Щоб вставити тунель, просто видаліть елемент на

дочірній сторінці, а потім CORE автоматично вставить тунель у верхню частину стрілки на батьківській сторінці.

Модель IDEF0 повинна мати мету та точку зору. Зазвичай вони розміщуються на контекстній сторінці. У CORE їх можна вставити за допомогою команди «Вставити примітку».

Призначення — короткий виклад причини існування моделі.

Точка огляду — короткий виклад ракурсу моделі.

Основними елементами діаграми в нотації IDEF0 є:

– блоки, у вигляді яких представлені процеси, функції, операції, дії (в залежності від ступеня деталізації);

– стрілки, у вигляді яких на діаграмі відображають інформаційні та матеріальні ресурси, пов'язані з функціями.

На рисунку 3.1 зображено контексну діаграму IDEF0, головним процесом якої є обробка зображення, а також зображені входи, виходи, механізми та ресурси.

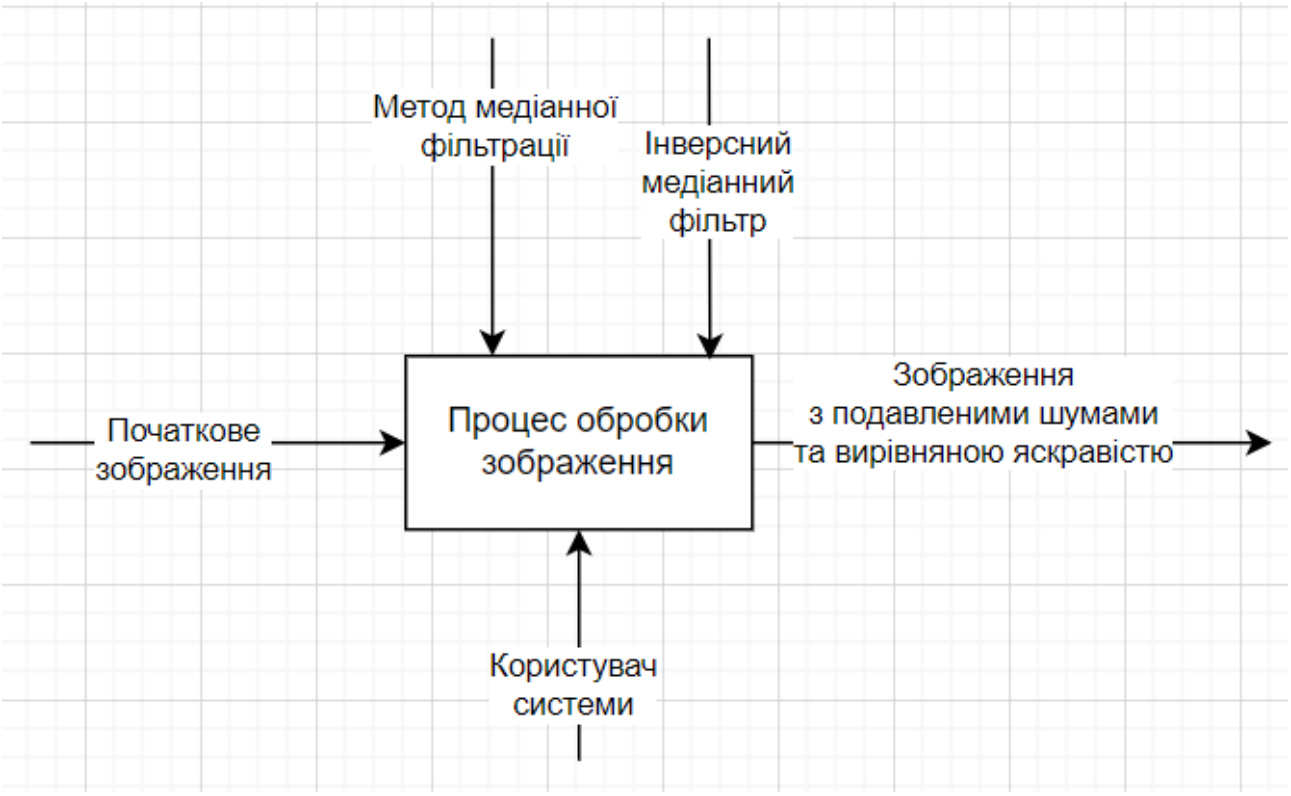

Рисунок 3.1 – IDEF0 контексна діаграма системи

3.2 Аналіз варіантів використання

При проектуванні функціонального навантаження системи слід проаналізувати використані варіанти, що дасть повне розуміння призначення та можливої діяльності інформаційної системи в майбутньому.

Найпростіша діаграма використання — це спосіб мислення про взаємодію між користувачем і системою, який показує відносини між користувачем і різними видами використання, в яких бере участь користувач. Діаграми використання можуть ідентифікувати різні типи користувачів системи та різні моделі використання, і часто супроводжуються діаграмами інших типів. Ціль зображується колом або еліпсом.

Хоча кожну можливість можна детально розглянути за допомогою самих параметрів, використання діаграм може допомогти надати огляд системи вищого рівня. Я вже говорив: «План використання — це принцип вашої системи».

Завдяки своїй спрощеній природі плани використання можуть бути хорошим інструментом комунікації для зацікавлених сторін. Ці малюнки намагаються змоделювати реальний світ і дозволити зацікавленим сторонам зрозуміти, як буде розроблена система. Сіау і Лі провели дослідження, щоб визначити, чи існують реальні сценарії використання чи не потрібні. Було виявлено, що використання діаграм передає намір системи зацікавленим сторонам більш спрощеним чином, і вони «повніше пояснюються, ніж діаграми класів».

Метою використання діаграм є показати динамічні аспекти системи. Для забезпечення повного функціонального та технічного представлення системи можна використовувати додаткові схеми та документи. Вони забезпечують спрощене та графічне представлення того, що насправді має робити система.

– Межа системи - прямокутник з ім'ям і еліпсом (прецедент). За відсутності корисної інформації її зазвичай можна опустити,

– Актор (англ. actor) — стилізована роль людини, яка являє собою набір ролей користувачів, які взаємодіють із сутностями (системами, підсистемами,

класами) (широке розуміння: люди, зовнішні сутності, класи, інші системи). Актори не можуть бути пов'язані один з одним (за винятком відносин обробки/дослідження),

– Прецедент - еліпс з написом, що вказує на виконану системну операцію (можливо, включаючи можливі варіанти), що привела до результатів, які спостерігає учасник. Назва може бути назвою або описом (з точки зору актора) системи «що» (а не «як»). Під час сценарію актори обмінюються системною інформацією. Сценарій можна відобразити в прецедентній діаграмі відео коментарів UML. Кілька різних сценаріїв можуть бути пов’язані з прецедентом.

На рисунку 3.2 зображено діаграму варіантів використання, яка описує можливі дії користувача в системі.

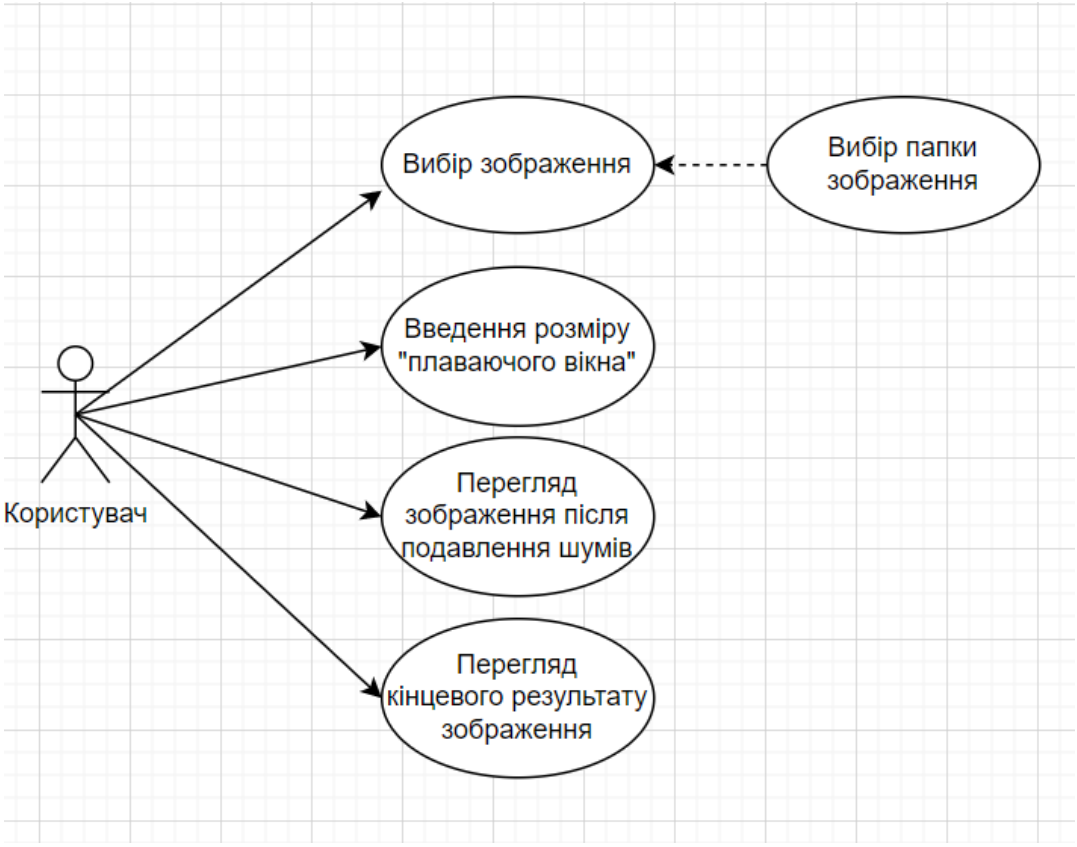

Рисунок 3.2 – Діаграма варіантів використання

3.3 Побудова діаграми діяльності системи

Діаграма діяльності є ще однією важливою діаграмою в UML для опису динамічних аспектів системи.

Діаграма діяльності — це, в основному, блок-схема, яка відображає потік від однієї діяльності до іншої. Діяльність можна описати як роботу системи.

Потік керування переходить від однієї операції до іншої. Цей потік може бути послідовним, розгалуженим або одночасним. Діаграми діяльності стосуються всіх типів керування потоком за допомогою різних елементів, таких як fork, join тощо.

Основні цілі діаграм діяльності подібні до інших чотирьох діаграм. Він фіксує динамічну поведінку системи. Інші чотири діаграми використовуються для показу потоку повідомлень від одного об’єкта до іншого, але діаграма діяльності використовується для показу потоку повідомлень від однієї діяльності до іншої.

Діяльність – це певна операція системи. Діаграми діяльності використовуються не лише для візуалізації динамічної природи системи, але й для побудови виконуваної системи за допомогою методів прямого та зворотного проектування. Єдине, чого не вистачає на діаграмі активності – це частина повідомлення.

Він не показує жодного потоку повідомлень від однієї діяльності до іншої. Діаграму діяльності іноді розглядають як блок-схему. Хоча діаграми виглядають як блок-схеми, вони не є такими. Він показує різні потоки, такі як паралельні, розгалужені, одночасні та одиночні.

Мета діаграми діяльності може бути описана як:

– намалювати потік діяльносі системи.

– описати послідовність від однієї діяльності до іншої.

– описати паралельний, розгалужений і одночасний потік системи.

На рисунку 3.3 зображено діаграму діяльності системи, яка описує потік роботи та динамічну поведінку системи.

Основними діяльностями є завантажання зображення, старт та проведення подавлення шумів за допомогою методу медіанної фільтрації, старт за проведення вирівнювання яскравості за допомогою Фур’є аналізу та видача кінцевого результату.

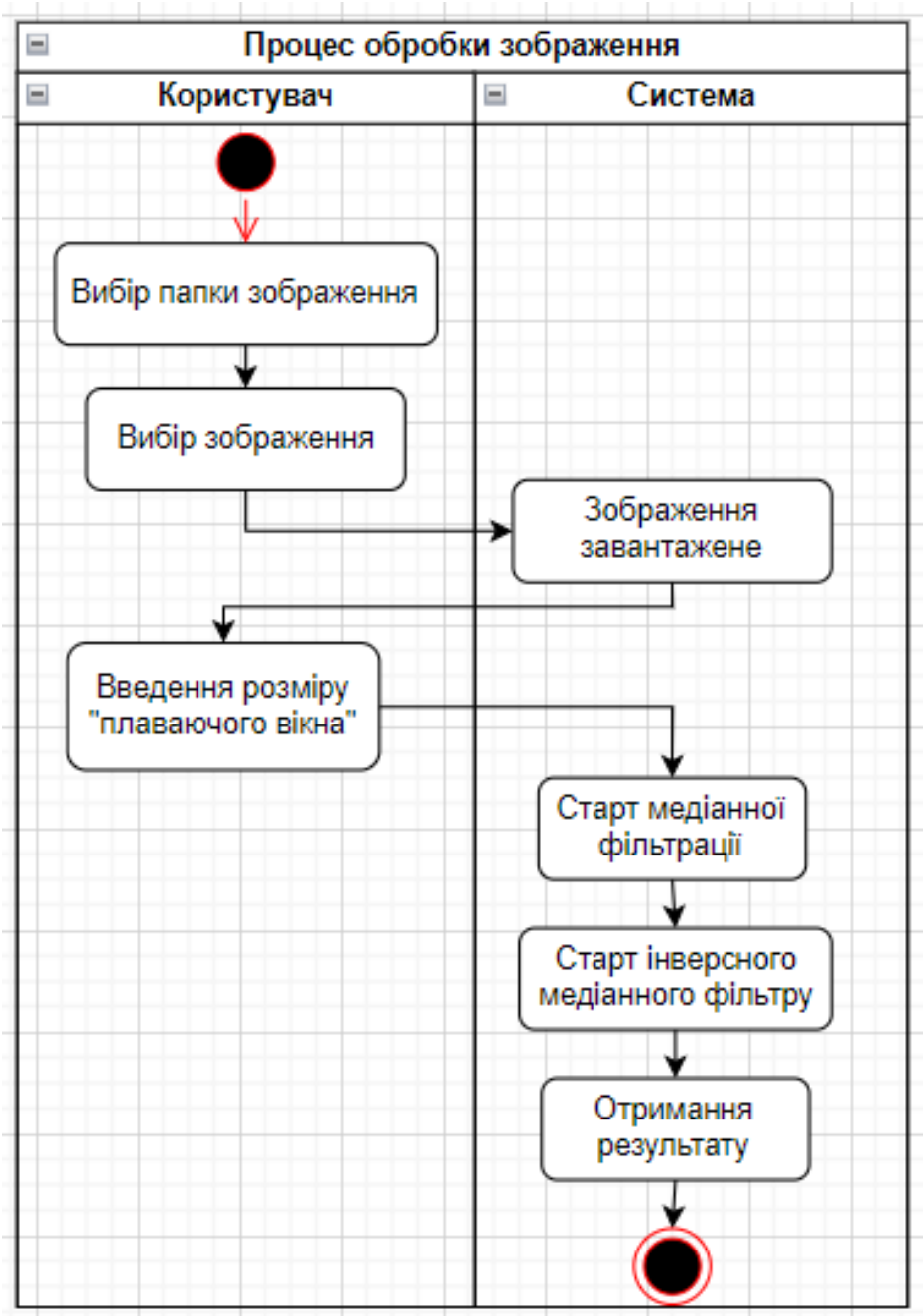

Рисунок 3.3 – Діаграма діяльності системи

3.4 Проектування внутрішньої будови системи за допомогою діаграми класів

При проектуванні програмного забезпечення внутрішня структура системи зазвичай відображається у вигляді діаграми класів, яка показує структуру класів і модулів проекту та взаємодію між ними.

У програмній інженерії діаграма класів уніфікованої мови моделювання (UML) — це статична структурна діаграма, яка описує структуру системи та показує системні класи, їх атрибути, операції (або методи) і зв’язки між об’єктами.

Діаграми класів є основними будівельними блоками об'єктно-орієнтованого моделювання. Використовується для загального концептуального моделювання структури програми та детального моделювання для перетворення моделі в програмний код. Діаграми класів також можна використовувати для моделювання даних. Класи на діаграмі класів представляють основні елементи, взаємодії в програмі та класи, що програмуються.

На малюнку клас представлений вікном, що містить три відділення:

– У верхньому відділенні міститься назва класу. Надруковано жирним шрифтом і відцентровано, а перша літера написана великими літерами.

– Середній відсік містить атрибути класу. Вони вирівняні за лівим краєм, а перша буква мала.

– У нижньому відділенні містяться операції, які може виконувати клас.

Вони також вирівняні по лівому краю, а перша літера є малою.

При проектуванні системи визначення багатьох класів і групування їх у схему класів допомагає визначити статичні відносини між ними. При детальному моделюванні категорія концептуального проектування зазвичай поділяється на кілька підкатегорій.

Залежність — це смислове відношення між залежними елементами та незалежними елементами моделі. Якщо зміна визначення одного елемента (сервера чи цілі) може призвести до зміни іншого (клієнта чи джерела), він існує

між двома елементами. Це об’єднання одностороннє. Релевантність відображається пунктирною лінією з відкритою стрілкою, що вказує від замовника до постачальника.

Для подальшого опису поведінки системи ці діаграми класів можуть бути доповнені діаграмами станів або автоматами станів UML.

Асоціація являє собою серію посилань. Бінарні асоціації (з двома кінцями) зазвичай виражаються у вигляді рядків. Асоціація може пов'язувати будь-яку кількість класів. Асоціація з трьома ланками називається потрійною асоціацією. Асоціацію можна назвати, а кінець асоціації можна змінити за допомогою таких атрибутів, як назва ролі, індикатор власності, множинність і видимість.

Існує чотири різні типи асоціацій: двостороння, одностороння, агрегаційна (включаючи комбіновану агрегацію) та рефлексивна. Найбільш поширеними є двосторонні та односторонні асоціації.

Наприклад, клас польоту асоціюється з класом літака з двостороннім рухом. Асоціація являє собою статичні відносини, спільні між об'єктами двох класів.

Агрегація є варіантом відношення "mas"; агрегація є більш специфічною, ніж асоціація. Це асоціація, яка представляє частину мети або частину відносин. Як показано на малюнку, у професора «має» клас для викладання. Як тип асоціації, агрегація може бути названа й мати ті самі модифікації, що й асоціація. Однак агрегація не може включати більше двох категорій; це має бути бінарна асоціація. Крім того, майже немає різниці між агрегацією та асоціацією в процесі впровадження, і графік може повністю опустити зв’язок агрегації. [7]

Коли клас є колекцією або контейнером інших класів, відбувається агрегація, але класи, що містяться, не мають сильної залежності від життєвого циклу контейнера. Коли контейнер знищено, вміст контейнера все ще існує.

В UML він представлений графічно у вигляді порожнистого ромба на класі хоста, з’єднавши його з однією лінією класу хоста. Агрегація - це семантично розширений об'єкт, який розглядається як одиниця в багатьох операціях, хоча фізично складається з кількох менших об'єктів.

Приклад: бібліотека та студенти. Тут студенти можуть існувати без бібліотеки, а зв’язок між студентами та бібліотекою є сукупністю.

Це показує, що один із двох споріднених класів (підкласів) вважається спеціалізацією іншого (супертипу), а суперклас є узагальненням підкласів. На практиці це означає, що будь-який екземпляр підтипу також є екземпляром супертипу. Типове дерево узагальнень цієї форми можна знайти в біологічній класифікації: люди — це підклас вищих приматів, це підклас ссавців тощо. Цей зв’язок найлегше зрозуміти як фразу «А є Б» (люди — це ссавці, а ссавці — тварини).

Графічне представлення узагальнення UML — це форма порожнистого трикутника в кінці суперкласу рядків (або дерева рядків), що з’єднує його з одним або кількома підтипами.

Відношення узагальнення також називають спадковістю або відносинами «є».

Суперклас (базовий клас) у відносинах узагальнення також називають «батьківським класом», суперкласом, базовим класом або базовим типом.

Підтипи у відносинах спеціалізації також називаються "дочірніми" підкласами, похідними класами, похідними типами, успадкованими класами або успадкованими типами.

Зауважте, що ці відносини повністю відрізняються від біологічних відносин між батьком і дитиною: використання цих термінів дуже поширене, але може ввести в оману.

А є типом В. Наприклад, «дуб — це різновид дерева», «автомобіль — це вид транспорту». Підсумок може відображатися лише в діаграмах класів і діаграмах використання.

У моделюванні UML відносини реалізації — це відносини між двома елементами моделі, де один елемент моделі (клієнт) реалізує (реалізує або виконує) поведінку, задану іншим елементом моделі (постачальником).

Графічне представлення реалізації UML являє собою порожнистий трикутник, з’єднаний з кінцем штрихового інтерфейсу (або дерева рядків)

одного або кількох реалізаторів. Проста стрілка використовується в кінці пунктирного інтерфейсу, який з’єднує його з користувачем. На діаграмі компонентів використовується графічна умова «м'ячний розетка» (реалізатор кладе кульку або льодяник, а користувач показує кульку). Реалізація може бути відображена лише на діаграмі класів або компонентів. Реалізація – це зв’язок між класами, інтерфейсами, компонентами та пакетами, які з’єднують елементи замовника з елементами постачальника. Відношення реалізації між класом/компонентом та інтерфейсом показує, що клас/компонент реалізує операції, запропоновані інтерфейсом.

Залежність — це слабкіша форма спілкування, яка вказує на те, що один клас залежить від іншого класу, оскільки він використовує його в певний момент часу. Якщо незалежний клас залежить від змінної параметра або локальної змінної методу, то один клас залежить від іншого. Це відрізняється від асоціації, де атрибут залежного класу є екземпляром незалежного класу. Іноді відносини між цими двома класами слабкі. Вони взагалі не реалізуються зі змінними-членами. Натомість вони можуть бути реалізовані як параметри функцій-членів.

Представлення асоціації UML — це лінія, що з'єднує два пов'язані класи. На кожному кінці рядка є додаткові символи. Наприклад, ми можемо використовувати стрілку, щоб вказати, що кінчик можна побачити з хвоста стрілки. Ми можемо представити право власності, помістивши кульку, і роль, яку відіграє елемент на цьому кінці, полягає в вказуванні імені ролі та кількох екземплярів сутності (діапазон об’єктів, пов’язаних з іншого кінця).

Класи сутності моделюють довгоживучу інформацію, якою обробляє система, а іноді і поведінку, пов'язану з цією інформацією. Їх не слід ідентифікувати як таблиці баз даних чи інших сховищ даних.

Вони намальовані як кола з короткою лінією, прикріпленою до нижньої частини кола. Як варіант, їх можна намалювати як звичайні класи із позначенням стереотипу «сутність» над назвою класу.

На рисунку 3.2 зображено діаграму класів, яка відображає внутрішню будову проекту.

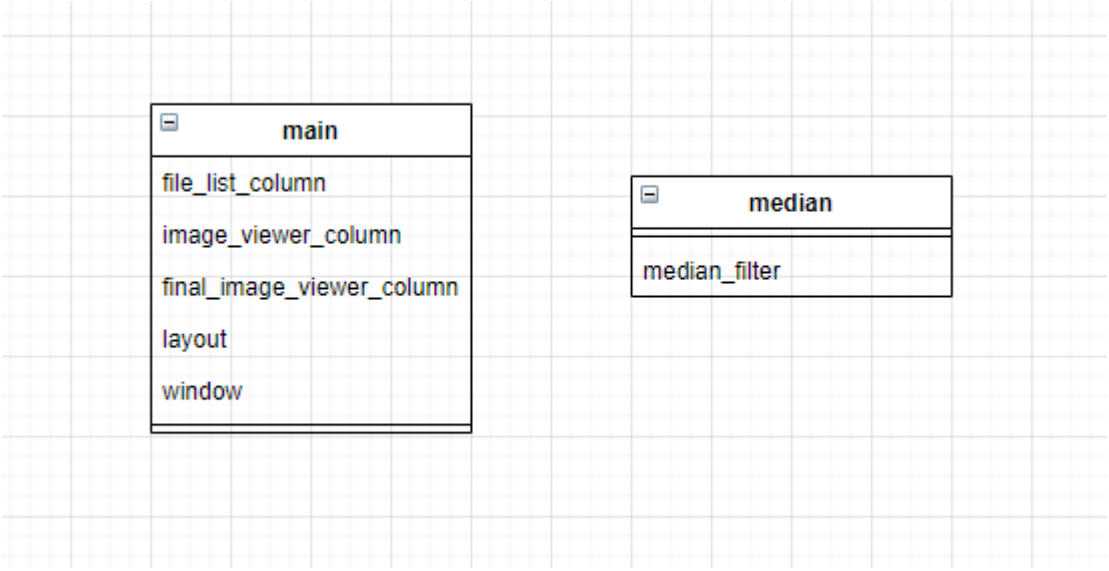

Рисунок 3.4 – Діаграма класів системи

3.5 Розробка інтерфейсу користувача

GUI — графічний інтерфейс користувача — це форма інтерфейсу користувача, яка дозволяє користувачам взаємодіяти з електронними пристроями за допомогою графічних піктограм і аудіоіндикаторів, таких як первинна нотація, замість текстових інтерфейсів користувача, введених міток команд або текстової навігації. Графічні інтерфейси користувача були введені у відповідь на сприйману стрімку криву навчання CLI (інтерфейсів командного рядка),[3][4][5] які вимагають введення команд на клавіатурі комп'ютера.

Дії в GUI зазвичай виконуються через пряме маніпулювання графічними елементами.[6][7][8] Окрім комп'ютерів, GUI використовуються в багатьох кишенькових мобільних пристроях, таких як MP3-плеєри, портативні медіа-плеєри, ігрові пристрої, смартфони та менші побутові, офісні та промислові засоби керування. Термін GUI, як правило, не застосовується до інших типів інтерфейсів з нижчою роздільною здатністю дисплея, таких як відеоігри (де перевага віддається HUD (дисплею з головою) [9], або не включає плоскі екрани, такі як об'ємні дисплеї [10], оскільки термін обмежується сферою застосування

екранів двовимірних дисплеїв, здатних описувати загальну інформацію, відповідно до традиції досліджень інформатики в Дослідницькому центрі Xerox Palo Alto.

Розробка візуальної композиції та тимчасової поведінки графічного інтерфейсу є важливою частиною програмування прикладних програм у сфері взаємодії людини з комп'ютером. Його мета полягає в тому, щоб підвищити ефективність і простоту використання для основного логічного дизайну збереженої програми, дисципліни дизайну під назвою юзабіліті. Методи дизайну, орієнтованого на користувача, використовуються, щоб гарантувати, що візуальна мова, введена в дизайн, добре адаптована до завдань.

Функції видимого графічного інтерфейсу програми іноді називають chrome або GUI (вимовляється «липкий»). [11][12][13] Як правило, користувачі взаємодіють з інформацією, маніпулюючи візуальними віджетами, які дозволяють взаємодіяти відповідно до типу даних, які вони зберігають. Віджети добре розробленого інтерфейсу підібрані для підтримки дій, необхідних для досягнення цілей користувачів. Модель–подання–контролер дозволяє створювати гнучкі структури, в яких інтерфейс незалежний від функцій програми та опосередковано пов'язаний із ними, тому графічний інтерфейс можна легко налаштувати. Це дозволяє користувачам вибирати або створювати іншу оболонку за бажанням і полегшує роботу дизайнера щодо зміни інтерфейсу в міру розвитку потреб користувача. Хороший дизайн графічного інтерфейсу більше стосується користувачів, а менше – архітектури системи. Великі віджети, такі як вікна, зазвичай забезпечують рамку або контейнер для основного вмісту презентації, такого як веб-сторінка, повідомлення електронної пошти або малюнок. Менші зазвичай діють як інструмент для введення користувачем.

Графічний інтерфейс користувача може бути розроблений відповідно до вимог вертикального ринку як графічний інтерфейс користувача для конкретної програми. Приклади включають банкомати (АТМ), сенсорні екрани торгових точок (POS) у ресторанах [14], каси самообслуговування, які використовуються в роздрібних магазинах, самостійне оформлення квитків у авіакомпаніях і

реєстрація на рейс, інформаційні кіоски в громадських місцях, наприклад залізничний вокзал або музей, а також монітори або екрани керування у вбудованих промислових програмах, які використовують операційну систему реального часу (RTOS).

Стільникові телефони та кишенькові ігрові системи також використовують графічний інтерфейс із сенсорним екраном для спеціальних програм. Новіші автомобілі використовують GUI у своїх навігаційних системах і мультимедійних центрах або комбінаціях навігаційних мультимедійних центрів.

В даному випадку інтерфейс користувача складається з одного вікна, на якому відображається первинний і кінцевий результат роботи системи, вибір зображення і зерно фільтрації.

Користувач має змогу загрузити бажане зображення у програму, далі ввести розмір «плаваючого» вікна у пікселях. Після цього був застосований медіанний фільтр для того, щоб прибрати шуми та його інверсія, за допомогою якої було вирівнено яскравість зображення. Після цього користувач має змогу побачити кінцевий результат зображення.

Вікно програми показано на рисунку 3.3.

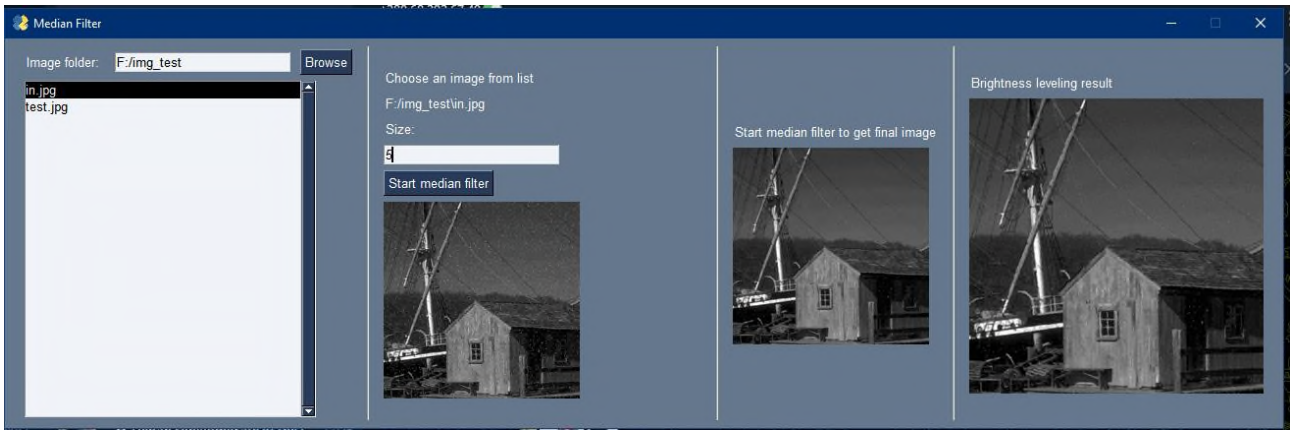

Рисунок 3.5 — Графічний інтерфейс системи

3.6 Тестування системи

Тестування програмного забезпечення – це акт перевірки артефактів і поведінки програмного забезпечення, що тестується, шляхом перевірки та верифікації. Тестування програмного забезпечення також може забезпечити об’єктивне, незалежне уявлення про програмне забезпечення, щоб дозволити бізнесу оцінити та зрозуміти ризики впровадження програмного забезпечення. Методи тестування включають, але не обов’язково обмежуються:

– аналіз вимог до продукту щодо повноти та правильності в різних контекстах, таких як галузева перспектива, бізнес-перспектива, доцільність та життєздатність впровадження, зручність використання, продуктивність, безпека, міркування інфраструктури тощо.

– перегляд архітектури продукту та загального дизайну продукту

– робота з розробниками продуктів над удосконаленням техніки кодування, шаблонів проектування, тестів, які можна написати як частину коду на основі різних методів, таких як граничні умови тощо.

– виконання програми або програми з метою перевірки поведінки

– перевірка інфраструктури розгортання та пов’язаних скриптів та автоматизації

– брати участь у виробничій діяльності, використовуючи методи моніторингу та спостереження

Тестування програмного забезпечення може надати користувачам або спонсорам об’єктивну, незалежну інформацію про якість програмного забезпечення та ризик його збою.

Несправності та збої. Помилки програмного забезпечення виникають через наступний процес: Програміст робить помилку (помилку), що призводить до помилки (дефекту, помилки) у вихідному коді програмного забезпечення. Якщо ця помилка виконана, у певних ситуаціях система дасть неправильні результати, що призведе до збою.

Не всі несправності обов'язково призведуть до збоїв. Наприклад, помилки в мертвому коді ніколи не призведуть до збоїв. Несправність, яка не виявила збоїв, може призвести до збою при зміні середовища. Приклади цих змін у середовищі включають програмне забезпечення, яке запускається на новій апаратній платформі комп’ютера, зміни вихідних даних або взаємодію з іншим програмним забезпеченням. Одна несправність може призвести до широкого спектру симптомів відмови.

Не всі помилки програмного забезпечення викликані помилками кодування. Одним із поширених джерел дорогих дефектів є прогалини у вимогах, тобто нерозпізнані вимоги, які призводять до помилок, пропущених розробником програми. Прогалини вимог часто можуть бути нефункціональними вимогами, такими як тестованість, масштабованість, ремонтопридатність, продуктивність та інші вимоги. безпеки.

Статичне, динамічне та пасивне тестування. У тестуванні програмного забезпечення існує багато підходів. Огляди, покрокові інструкції або перевірки називаються статичним тестуванням, тоді як виконання запрограмованого коду з заданим набором тестових випадків називається динамічним тестуванням.

Статичне тестування часто є неявним, як-от коректура, а також коли інструменти програмування/текстові редактори перевіряють структуру вихідного коду, а компілятори (попередні компілятори) перевіряють синтаксис і потік даних як статичний аналіз програми. Динамічне тестування відбувається під час запуску самої програми. Динамічне тестування може розпочатися до того, як програма буде на 100% завершена, щоб перевірити окремі розділи коду та застосувати до дискретних функцій або модулів. Типовими методами для них є або використання заглушок/драйверів, або виконання з середовища налагодження.

Статичне тестування передбачає перевірку, тоді як динамічне також передбачає перевірку. Пасивне тестування означає перевірку поведінки системи без будь-якої взаємодії з програмним продуктом. На відміну від активного тестування, тестувальники не надають жодних тестових даних, а переглядають

системні журнали та трасування. Вони шукають моделі та специфічну поведінку, щоб приймати якісь рішення. Це пов’язано з перевіркою часу виконання в автономному режимі та аналізом журналу.

Дослідницький підхід. Дослідницьке тестування — це підхід до тестування програмного забезпечення, який коротко описується як одночасне навчання, розробка тесту та виконання тесту. Джем Канер, який ввів цей термін у 1984 році,:2 визначає дослідницьке тестування як «стиль тестування програмного забезпечення, який підкреслює особисту свободу та відповідальність окремого тестувальника за постійну оптимізацію якості своєї роботи, розглядаючи тест-пов’язане навчання, дизайн тесту, виконання тесту та інтерпретація результатів тесту як взаємодопоміжні дії, які виконуються паралельно протягом усього проекту».

«Коробковий» підхід. Методи тестування програмного забезпечення традиційно поділяються на тестування білого та чорного ящиків. Ці два підходи використовуються для опису точки зору, яку дотримується тестувальник під час розробки тестових випадків. До методології тестування програмного забезпечення також можна застосувати гібридний підхід, який називається тестуванням сірого ящика. Оскільки концепція тестування сірого ящика, яка розробляє тести на основі конкретних елементів дизайну, стає все більш популярною, це «довільне розходження» між тестуванням чорного та білого ящиків дещо зникло.

Таблиця 3.1 — Тестування додатку

№	Тест-кейс	Очікуваний результат	Отриманий результат
1	Запуск без зображення	Система сповіщає про відсутність зображення	Система сповіщає про відсутність зображення
2	Запуск з іншим файлом замість зображення	Система сповіщає про помилку читання зображення	Система сповіщає про помилку читання зображення

Результати тестування показують, що система повністю функціональна і готова до використання в реальних умовах

ВИСНОВКИ

Мета роботи полягала в створенні системи для автоматичного вирівнювання яскравості і контрастності зображень.

Отже, у результаті виконання кваліфікаційної роботи вирішені наступні завдання:

– проведений аналіз предметної області обробки зображень;

– проведений огляд поняття вирівнювання яскравості;

– обраний методи обробки зображень для подальшого засосування в програмі;

– проведений аналіз методів та засобів розробки ПЗ;

– обрано мову програмування;

– обрано додаткові інструменти;

– обрано середовище розробки;

– проаналізовано концепцію системи та провести її проектування;

– проведено аналіз варіантів використання;

– розроблено графічний інтерфейс користувача;

– проведено тестування системи.

Завдяки чіткому виконанню завдань, поставлених на початку роботи, в результаті отримано повноцінну програму вирівнювання яскравості зображень, яка готова до використання в реальних умовах.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. ДСТУ 3008:2015. Інформація та документація. Звіти у сфері науки і техніки. Структура та правила оформлювання. Чинний від 2017-07-01. – Київ: ДП «УкрНДНЦ», 2016. – 26 с.

2. Фаулер М. UML. Основы, 3е издание. – Пер. с англ. – СПб: СимволПлюс, 2004. – 192 с., ил.

3. "application software". Oxford English Dictionary (Online ed.). Oxford University Press. (Subscription or participating institution membership required.)

4. R. Shirey (August 2007). Internet Security Glossary, Version 2. Network Working Group. doi:10.17487/RFC4949. RFC 4949. Informational.

5. "Application software". PC Magazine. Ziff Davis.

6. Ryan, Thorne (2013-03-14). "Caffeine and computer screens: student programmers endure weekend long appathon". The Arbiter. Archived from the original on 2016-07-09. Retrieved 2015-10-12.

7. Ceruzzi, Paul E. (2000). A History of Modern Computing. Cambridge, Massachusetts: MIT Press. ISBN 0-262-03255-4.

8. Ulrich, William. "Application Package Software: The Promise Vs. Reality". Cutter Consortium.

9. Application Package Software: The Promise Vs. Reality

10. The History of 'App' and the Demise of the Programmer

11. Gassée, Jean-Louis (2012-09-17). "The Silly Web vs. Native Apps Debate". Archived from the original on 2016-04-15. Retrieved 2013-07-14.

12. Frechette, Casey (2013-04-11). "What journalists need to know about the difference between Web apps and native apps". Poynter. Retrieved 2017-01-04.

13. Valums, Andrew (2010-02-10). "Web apps vs desktop apps". valums.com. Archived from the original on 2013-04-02. Retrieved 2013-07-14.

14. "What Is a Horizontal Application?".

15. "What Are Horizontal Services?". Archived from the original on 2013-10-31.

16. "What is Application Software & Its Types | eduCBA". eduCBA. 2015-12-21. Retrieved 2017-03-24.

17. Campbell-Kelly, Martin; Aspray, William (1996). Computer: A History of the Information Machine. New York: Basic Books. ISBN 0-465-02990-6.

18. "Definition of desktop application". PCMAG. Retrieved 2022-01-07.

19. "Application Development (AppDev) Defined and Explained". Bestpricecomputers.co.uk. 13 August 2007. Retrieved 5 August 2012.

20. DRM Associates (2002). "New Product Development Glossary". Retrieved 29 October 2006.

21. System Development Methodologies for Web-Enabled E-Business: A Customization Framework Linda V. Knight (DePaul University, USA), Theresa A. Steinbach (DePaul University, USA) and Vince Kellen (Blue Wolf, USA)

22. Joseph M. Morris (2001). Software Industry Accounting. p.1.10

23. Alan M. Davis. Great Software Debates (October 8, 2004), pp:125-128 Wiley-IEEE Computer Society Press

24. Otero, Carlos. "Software Design Challenges". IT Performance Improvement. Taylor & Francis LLC. Retrieved 19 October 2017.

25. Edward J. Barkmeyer ea (2003). Concepts for Automating Systems Integration NIST 2003.

26. Paul R. Smith & Richard Sarfaty (1993). Creating a strategic plan for configuration management using Computer Aided Software Engineering (CASE) tools. Paper For 1993 National DOE/Contractors and Facilities CAD/CAE User's Group.

27. Kuhn, D.L (1989). "Selecting and effectively using a computer-aided software engineering tool". Annual Westinghouse computer symposium; 6–7 Nov 1989; Pittsburgh, PA (USA); DOE Project.

28. P. Loucopoulos and V. Karakostas (1995). System Requirements Engineering. McGraw-Hill.

29. CASE Archived 2012-02-18 at the Wayback Machine definition In: Telecom Glossary 2000 Archived 2005-11-22 at the Wayback Machine. Retrieved 26 Oct 2008.

30. K. Robinson (1992). Putting the Software Engineering into CASE. New York : John Wiley and Sons Inc.

31. Xiao He (2007). "A metamodel for the notation of graphical modeling languages". In: Computer Software and Applications Conference, 2007. COMPSAC 2007 – Vol. 1. 31st Annual International, Volume 1, Issue, 24–27 July 2007, pp 219–224.

32. Merx, Georges G.; Norman, Ronald J. (2006). Unified Software Engineering with Java. Prentice-Hall, Inc. p. 201. ISBN 0130473766.