Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Факультет інформатики
Кафедра інформатики

Автоматизоване тестування адаптивності інтерфейсу iOS/iPadOS додатків

Текстова частина до курсової роботи
за спеціальністю «Інженерія програмного забезпечення» - 121

Виконав: студент 3-го року навчання,

Освітньої програми «Інженерія програмного забезпечення», 121

Гетьман Дмитро Андрійович

Керівник Франків О.О.,_________
асистент

Рецензент ________________________
(прізвище та ініціали)

Секретар ЕК
____
«____» __________________ 20____ р.

Київ – 2023

Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Факультет інформатики
Кафедра інформатики

ЗАТВЕРДЖУЮ
Викладач кафедри інформатики ,
канд. фіз-мат. наук, доц. ____________ Гороховський С.С.
(підпис)
„____”___________________________2023р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ
на курсову роботу
студенту Гетьману Дмитру Андрійовичу
факультету інформатики 3 курсу бакалаврської програми
ТЕМА: Автоматизоване тестування адаптивності інтерфейсу iOS/iPadOS додатків

Зміст ТЧ до курсової роботи:
- Індивідуальне завдання
- Вступ
- Огляд теоретичного матеріалу і здійснення дослідження
- Висновки
- Список посилань

Дата видачі „___” _________ 2022 р. Керівник_______________
(підпис)
Завдання отримав __________
(підпис)

Тема: **Автоматизоване тестування адаптивності інтерфейсу iOS/iPadOS додатків**

№ п/п	Назва етапу курсової роботи	Термін виконання етапу	Примітка
1.	Отримання завдання на курсову роботу.	17.10.2022	
2.	Огляд документації за темою роботи.	18.11. 2023	
3.	Проведення досліджень.	25.02. 2023	
3.	Опис результатів дослідження.	03.04. 2023	
4.	Аналіз отриманих результатів з керівником.	18.04. 2023	
6.	Корегування роботи.	25.04. 2023	
7.	Створення презентації та написання доповіді.	01.05. 2023	
8.	Остаточне оформлення пояснювальної роботи та слайдів.	15.05. 2023	
9.	Захист курсової роботи.	24.05. 2023	

Студент ______________________________________

Керівник _____________________________________

“______” ________

Зміст

АНОТАЦІЯ 6
ВСТУП 7
РОЗДІЛ 1. АВТОМАТИЗОВАНЕ ТЕСТУВАННЯ МОБІЛЬНИХ ДОДАТКІВ 9
1.1 Означення та принцип роботи тестування мобільних додатків 9
1.2 Особливості автоматизованого тестування мобільних додатків.. 12
1.3 Агенти у контексті автоматизованого тестування................ 13
1.4 Метод використання засобів OpenCV з реалізацією на Python 16
1.5 Об’єднання засобів XCTest та OpenCV 18
1.6 Висновки до розділу 1 19
РОЗДІЛ 2. Обґрунтування підходу до вирішення проблеми автоматизованого тестування адаптивності інтерфейсу на пристроях iOS/iPadOS 21
2.1 Підхід обраний для реалізації автоматизованої системи 21
2.2 Обґрунтування способу реалізації....................................... 23
2.3 Обґрунтування вибору фреймворку OpenCV для тестування інтерфейсу 27
2.4 Порівняння системи з вже існуючими 28
2.5 Висновки до розділу 2 30
РОЗДІЛ 3. ОПИС ІМПЛЕМЕНТАЦІЇ ТА АНАЛІЗ РЕЗУЛЬТАТІВ............ 31
3.1 Опис імплементації системи порівняння скріншотів........... 31
3.2 Опис засобів застосованих для імплементації системи(XCTest) 32
3.3 Опис засобів застосованих для імплементації системи(OpenCV)... 34
3.4 Аналіз результатів 38
3.5 Висновки до розділу 3 40
ВИСНОВКИ.. 41

ДЖЕРЕЛА 43

АНОТАЦІЯ

В даній роботі розглянуто систему для автоматизованого тестування адаптивності інтерфейсу iOS/iPadOS додатків. Ця робота досліджує існуючі механізми використання XCTest та OpenCV для автоматизованого тестування інтерфейсу в мобільних застосунках на базі iOS. При цьому детально розглядається практичне застосування цих фреймворків, їхня поточна реалізація. До роботи також входить розробка нової програми, яка має на меті спростити впровадження автоматизованого тестування інтерфейсу, порівняння знімків екрана і опрацювання результатів для розробників мобільних додатків. Такий підхід допоможе полегшити створення нових додатків, а також вдосконалення уже існуючих, зокрема за рахунок більш точного виявлення проблем із дизайном інтерфейсу.

ВСТУП

Мобільні додатки стали невід'ємним елементом нашого повсякдення та мають дуже великий вплив на життя людства, тому коректна робота додатку впливає на досвід користувача та формує його ставлення до будь-якого програмного забезпечення, а саме долю, чи буде застосунок використовуватися чи ні. Для того, щоб уникнути випадків, коли додаток працює не так як задумано, під час розробки додатку існує етап тестування, під час якого виявляють несправності до моменту публікації версії з новими функціями. Завчасне уникнення можливих непередбачуваних поведінок додатку за допомогою тестування допоможе всій команді отримати негативних наслідків після надання користувачу погано відтестованого додатку.

Щоб подолати цю проблему, автоматизоване тестування стало потужною технікою, яка допоможе розробникам ефективно перевіряти адаптивність інтерфейсів додатків для iOS. Автоматизувавши процес тестування, розробники зможуть заощадити час і ресурси, гарантуючи при цьому, що їхні додатки відповідають найвищим стандартам продуктивності.

Метою цієї роботи є створення автоматизованої системи тестування адаптивності інтерфейсу iOS додатків. У роботі використаний метод порівняння знімків екрану телефону з очікуваним дизайном та отриманим актуальним, які отриманні після тестування додатку за допомогою використання можливостей фреймворку XCTest від Apple.

У першому розділі було розглянуто загальні відомості та означення автоматизованого тестування мобільних додатків, у випадку нашої проблеми це iOS/iPadOS додатки.

У другому розділі було досліджено, обґрунтування підходу до рішення проблеми, пояснення, чому саме існуючі методи у роботі було використані для вирішення проблеми.

У третьому розділі висвітлено опис імплементації розробленої автоматизованої системи тестування та аналіз результатів дослідження.

Було розроблено систему для тестування адаптивності інтерфейсу додатків на iOS/iPadOS, яка за певною базою скріншотів повідомляє розробника про несправності, наочно показує негативні результати тестування, а саме скріншоти з позначенням місць несправності інтерфейсу. Система дозволяє помірно зручно тестувати інтерфейс iOS додатків та її концепція може бути використана як фундамент для більш точної та розгалуженої системи тестування.

РОЗДІЛ 1. АВТОМАТИЗОВАНЕ ТЕСТУВАННЯ МОБІЛЬНИХ ДОДАТКІВ

1.1 Означення та принцип роботи тестування мобільних додатків

Тестування мобільних додатків є важливим процесом у життєвому циклі розробки програми(рис. 1.1), який гарантує, що функціональність, продуктивність, безпека та користувацький досвід відповідають очікуваним результатам. Тестування мобільних додатків передбачає оцінку програми на різних платформах, пристроях і мережевих конфігураціях для виявлення і виправлення помилок, вразливостей і проблем з зручністю використання. Деякі ключові аспекти тестування мобільних додатків включають тестування сумісності, тобто забезпечення безперебійної роботи програми на різних пристроях, операційних системах тощо. Тестування зручності використання, а саме вивчення користувацького інтерфейсу та загального досвіду користувача, щоб переконатися, що він інтуїтивно зрозумілий, простий у використанні та візуально привабливий. Також існує регресійне тестування, воно полягає в повторному тестуванні раніше протестованих компонентів після змін або оновлень, щоб переконатися, що не з'явилися нові проблеми.

Проводячи ретельне тестування мобільних додатків, розробники можуть виявити і вирішити проблеми до того, як додаток буде випущений, що в кінцевому підсумку призведе до створення більш якісного продукту і більш позитивного користувацького досвіду. Для більш зручного проведення процесу тестування QA спеціалісти створюють автоматизовані системи тестування, які допомагають збільшити продуктивність виконуваних тестувань додатків.

Рис 1.1. Життєвий цикл розробки програмного забезпечення

Принцип тестування мобільних додатків полягає у перевірці поведінки, продуктивності та сумісності програми на різних мобільних пристроях, операційних системах (таких як iOS і Android), мережевих умовах і сценаріях роботи користувачів. Дотримуючись цього принципу, тестування мобільних додатків спрямоване на досягнення наступних цілей:

- Функціональне тестування передбачає тестування основної функціональності мобільного додатку, щоб переконатися, що він відповідає заданим вимогам і поводиться так, як очікується. Тестувальники перевіряють функції, взаємодію з користувачем, навігацію, введення/виведення даних та обробку помилок, щоб переконатися, що додаток працює правильно.

- Тестування зручності використання фокусується на оцінці користувацького інтерфейсу (UI) та користувацького досвіду (UX) мобільного додатку. Тестувальники оцінюють інтуїтивність, простоту використання, візуальну привабливість та загальний шлях користувача. Мета полягає в тому, щоб користувачі могли легко взаємодіяти з додатком і отримували позитивний досвід.

- Тестування продуктивності має на меті оцінити чуйність, стабільність та ефективність мобільного додатку за різних умов навантаження. Тестувальники оцінюють такі фактори, як час запуску програми, час відгуку, використання пам'яті, споживання батареї, продуктивність мережі та здатність програми працювати з різними роздільними здатностями екрану. Мета - забезпечити оптимальну продуктивність і безперебійну роботу користувача.

- Тестування сумісності гарантує, що мобільний додаток буде коректно працювати на різних пристроях, з різними розмірами екранів, операційними системами та конфігураціями. Тестувальники проводять тести на різних реальних пристроях або емуляторах, щоб виявити та вирішити будь-які проблеми сумісності, які можуть виникнути. Це гарантує, що додаток буде доступним для широкого кола користувачів.

- Тестування безпеки фокусується на виявленні вразливостей, ризиків і потенційних порушень безпеки в мобільному додатку. Тестувальники оцінюють шифрування даних, механізми автентифікації, безпечний зв'язок і захист від поширених загроз безпеки. Мета - захистити дані користувачів і забезпечити безпеку додатку.

- Тестування локалізації гарантує, що мобільний додаток належним чином адаптований і перекладений для різних мов, культур і регіонів. Тестування інтернаціоналізації перевіряє, чи відповідає архітектура та дизайн додатку вимогам локалізації, таким як формати дат, валют і переклади мов. Це гарантує, що додаток може ефективно обслуговувати користувачів з різних місць.

- Тестування встановлення та оновлення, його назва відповідає сама за себе. На цьому етапі тестування перевіряється процес встановлення та оновлення мобільного додатку. Тестувальники перевіряють, чи можна без проблем встановити, оновити та видалити додаток. Також перевіряється цілісність даних, щоб переконатися, що дані користувача залишаються недоторканими під час процесу встановлення або оновлення.

- Регресійне тестування гарантує, що зміни або оновлення мобільного додатку не призведуть до появи нових помилок і не вплинуть на існуючу функціональність. Тестувальники повторно тестують раніше протестовані функції та сценарії, щоб підтвердити, що вони все ще працюють належним чином. Це гарантує, що додаток залишається стабільним і надійним протягом усього життєвого циклу.

Дотримуючись цих принципів і проводячи ретельне тестування мобільних додатків, розробники та організації можуть підвищити якість, надійність і задоволеність користувачів своїми мобільними додатками. Ефективні методи тестування допомагають виявити і вирішити проблеми на ранній стадії процесу розробки, що в кінцевому підсумку дозволяє отримати високоякісний мобільний додаток, який відповідає очікуванням користувачів.

1.2 Особливості автоматизованого тестування мобільних додатків

Автоматизоване тестування мобільних додатків - це процес, який передбачає використання інструментів, фреймворків і методів для виконання повторюваних і заздалегідь визначених дій над мобільними додатками з метою забезпечення їхньої функціональності, продуктивності, безпеки та загальної якості.

Автоматизувавши тести, розробники та команди QA можуть ефективно виявляти та виправляти помилки, регресії та інші проблеми перед тим, як випустити додаток кінцевим користувачам. Автоматизовані тестування мобільних додатків має немало переваг перед ручними аналогами основні з яких:

- підвищена точність досягається тим, що автоматизовані тести зменшують ймовірність людської помилки і можуть бути запущені кілька разів з послідовними результатами.
- підвищена ефективність досягається тим, що автоматизовані тести можна виконувати швидше, ніж ручні, що дозволяє командам тестувати більше сценаріїв за менший час.
- економічна ефективність проявляється в тому, що після налаштування автоматизованих тестів їх можна запускати багато разів без додаткових витрат, що може призвести до довгострокової економії.
- краще покриття тестів досягається тим, що автоматизоване тестування може охоплювати широкий спектр тестових кейсів і конфігурацій пристроїв, забезпечуючи кращу загальну якість програми.
- швидші цикли випуску, тобто швидко виявляючи та вирішуючи проблеми, команди можуть пришвидшити процес розробки та випуску.

Популярний інструмент та фреймворк для автоматизації тестування мобільних додатків, який будемо використовувати під час виконання завдання це XCTest. Цей інструменти підтримує різні типи тестування, але саме те, що нам потрібно це тестування інтерфейсу користувача на платформі iOS.

1.3 Агенти у контексті автоматизованого тестування

У нашому контексті автоматизованого тестування агентом виступає програмний компонент або інструмент, який взаємодіє з системою, що

тестується, імітуючи дії користувача та збираючи відповідні дані. Агенти відіграють вирішальну роль у виконанні тестових кейсів, моніторингу адаптивності інтерфейсу iOS додатків та генерації результатів тестування.

XCTest - це офіційний фреймворк від Apple для тестування інтерфейсу користувача для iOS додатків. Він надає надійний набір API та інструментів для автоматизації взаємодії з користувачем, перевірки елементів інтерфейсу та оцінки адаптивності інтерфейсів додатків для iOS. За допомогою XCTest можна імітувати жести, виконувати такі дії, як дотики та свайпи, а також перевіряти очікувану поведінку елементів інтерфейсу. XCTest виступає в ролі агента в процесі автоматизованого тестування, дозволяючи нам контролювати і відстежувати чуйність інтерфейсу iOS додатків. Наше використання цього фреймворку буде обмежене тільки генерацією скріншотів з різних частин програми та надання їх для наступного модуля програми, а саме порівння скріншотів з інтерфейсом по дизайну та існуючим, який ми отримали через XCTest.

XCTest діє як спритний виконавець, бездоганно повторюючи дії користувача, які він надав на виконання у процесі проведення тестів, і ретельно відстежуючи реакцію програми. Цей агент без особливих зусиль взаємодіє з інтерфейсом користувача, ретельно перевіряючи швидкість відгуку на кожному кроці. Завдяки точному контролю над додатком, XCTest дозволяє виконувати цільові тести, вимірювати час відгуку і перевіряти очікувану поведінку елементів інтерфейсу. Завдяки своїм діям XCTest дозволяє збирати цінні дані про продуктивність, які відображають чуйність інтерфейсу iOS додатків. Ці дані значно розширюють сферу можливостей для створення різнобічних методів тестування інтерфейсу мобільних додатків.

Так як XCTest це пряма частина розробки Apple, то ця система XCTest легко інтегрується з Xcode, бо є основним середовищем розробки додатків для iOS. Ця тісна інтеграція дозволяє легко писати, запускати та налагоджувати тести в одному середовищі розробки.

Якщо занурюватися трохи глибше можна прийти до розгляду базового класу XCTest – це XCTestCase. Як я вже зазначив XCTestCase - це базовий клас, що надається фреймворком XCTest від Apple для написання та виконання тестів при розробці додатків для iOS та macOS. XCTestCase діє як базовий клас для визначення окремих тестових випадків у наборі тестів.

Щоб створити тестовий кейс за допомогою XCTestCase, ми зазвичай створюємо підклас XCTestCase і визначаємо тестові методи в межах цього підкласу. Кожен метод тестування представляє окремий тестовий кейс, який ми хочемо виконати. Є певні методи тестування в XCTestCase, які мають певну угоду щодо іменування. Вони повинні починатися зі слова "test", за яким слідує описова назва, яка зазвичай відображає поведінку або властивість, що тестується. Наприклад, метод може мати назву "testScreenshotsCapturing" або "testDataParsing".

XCTestCase надає різноманітні методи тверджень, які дозволяють вам робити твердження про очікувану поведінку або результати під час виконання тестового кейсу. Ці твердження оцінюють умови і вказують, чи пройшов тест успішно, чи ні. Деякі поширені твердження включають XCTAssertEqual, XCTAssertTrue, XCTAssertNil тощо.

Щодо життєвий цикл тесту, то XCTestCase надає різні методи, які виконуються в певні моменти життєвого циклу тесту. Наприклад, методи setUp() і tearDown() викликаються до і після виконання кожного тестового методу, дозволяючи вам налаштувати і очистити будь-які необхідні ресурси тесту. Також XCTestCase дозволяє організовувати тестові кейси у набори тестів. Набір тестів - це колекція тестових кейсів, які можна виконувати разом. Набори тестів можна створити за допомогою методу allTests класу XCTestCase для визначення набору тестових кейсів, які потрібно включити до набору.

Існують засоби асинхронне тестування, тобто XCTestCase надає підтримку для тестування асинхронних операцій. Ми можемо використовувати об'єкти

XCTestExpectation для визначення очікувань, які будуть виконані асинхронно, що дозволяє вам тестувати асинхронний код і перевіряти виконання певних умов.

Також існують засоби для тестування продуктивності додатків. XCTestCase включає підтримку тестування продуктивності. Ми можемо виміряти і стверджувати продуктивність певних блоків коду або методів, використовуючи метрики продуктивності XCTest. Це дозволяє оцінити характеристики продуктивності нашого додатку при різних сценаріях.

За допомогою інструментів XCTestCase можемо робити запис даних. XCTestCase надає функціональність для запису та керування даними під час виконання тесту. Ми можемо записувати довільні дані за допомогою addAttachment(_:name:), що корисно для захоплення журналів, знімків екрану або будь-якої іншої важливої інформації під час тестування.

Доступна інтеграція з безперервною інтеграцією (CI). XCTestCase розроблений для роботи з системами безперервної інтеграції, такими як Jenkins, Travis CI або Xcode Server. Ми можемо легко інтегрувати тести на основі XCTest у свої робочі процеси CI, гарантуючи, що тести будуть автоматично виконуватися при внесенні змін до коду.

Використовуючи XCTestCase, розробники можуть створювати комплексні тестові кейси, виконувати твердження, обробляти асинхронні операції та вимірювати продуктивність у систематичний та організований спосіб. Це допомагає забезпечити якість і надійність додатків для iOS і macOS протягом усього процесу розробки.

1.4 Метод використання засобів OpenCV з реалізацією на Python

Перш ніж занурюватися в практичну частину, ми маємо з'ясувати деякі теоретичні аспекти, які напряму відносяться до роботи з бібліотекою OpenCV. Кожне зображення складається з різних пікселів. Піксель - це фундаментальний

елемент зображення. Можна уявити зображення як сітку, де кожна клітина цієї сітки містить один піксель, і верхній лівий кут зображення відповідає точці з координатами (0, 0). Наприклад, якщо у нас є зображення з роздільною здатністю 400x300 пікселів, це значить, що наша сітка має 400 рядків і 300 стовпців. Отже, в цьому зображенні є 400*300 = 120000 пікселів.

Зазвичай, пікселі зображень можна представити двома способами: у відтінках сірого і в RGB (червоний, зелений, синій) кольоровому просторі. В зображеннях у відтінках сірого кожен піксель має значення від 0 до 255, де 0 - це чорний колір, а 255 - білий. Значення між 0 і 255 представляють різні відтінки сірого, де ближчі до 0 значення темніші, а ближчі до 255 - світліші.

У кольоровому просторі RGB, кожен піксель складається з трьох компонентів: червоного, зеленого та синього. Кожна компонента має значення в діапазоні від 0 до 255, що показує "наскільки" цей колір присутній. Враховуючи, що кожна компонента має діапазон [0,255], нам потрібно лише 8-бітне беззнакове ціле число, щоб представити насиченість кольору. Потім ці три значення об'єднуються в кортеж типу (червоний, зелений, синій). Наприклад, для отримання білого кольору, кожна компонента має бути 255: (255, 255, 255). Навпаки, для отримання чорного кольору, кожна компонента має бути 0: (0, 0, 0).

Тож перейдемо до розгляду самої бібліотеки. OpenCV - це потужна бібліотека комп'ютерного зору, яка широко використовується в різних додатках, включаючи обробку та аналіз зображень. В контексті автоматизованого тестування OpenCV слугує додатковим агентом для аналізу та оцінки візуальних аспектів інтерфейсу iOS додатків. Ми можемо використовувати OpenCV в Python для створення скріншотів або відеозаписів інтерфейсу програми під час процесу тестування, але ця задача лежить на модулі у XCTests. Використовуючи можливості обробки зображень OpenCV2, ми можемо виділити відповідні візуальні особливості, проаналізувати покадрові зміни і виявити будь-які аномалії, які можуть вплинути на чуйність інтерфейсу. Частину описаного вище використовуємо у нашій роботі, тому можна буде побачити реальні приклади.

OpenCV, з його можливостями комп'ютерного зору, бере на себе роль далекоглядного спостерігача. Як додатковий агент, він розширює можливості оцінки, захоплюючи візуальні дані та аналізуючи складні деталі інтерфейсу. За допомогою OpenCV ми даємо можливість нашим агентам спостерігати за інтерфейсом в дії, аналізуючи знімки, які знаходяться у відповідних директоріях. Цей далекоглядний агент використовує методи обробки зображень для виявлення закономірностей, оцінки візуального зворотного зв'язку та виявлення будь-яких порушень, які можуть вплинути на невідповідність очікуваному інтерфейсу. Використовуючи OpenCV для аналізу, агенти розгадують нюанси, які можуть вплинути напряму на взаємодію з користувачем.

1.5 Об'єднання засобів XCTest та OpenCV

Інтеграція та об'єднання XCTest та OpenCV забезпечує комплексний підхід до автоматизованого тестування адаптивності інтерфейсу iOS-додатків. XCTest дозволяє автоматизувати взаємодію з користувачем і вимірювати показники продуктивності, що надаються фреймворком. У той же час OpenCV може доповнити аналіз, захоплюючи візуальні дані і виконуючи додаткові оцінки на основі зображень. Використовуючи OpenCV разом з XCTest, ми можемо аналізувати візуальні відгуки, оцінювати швидкість рендерингу елементів інтерфейсу та виявляти будь-які візуальні невідповідності або затримки, які можуть вплинути на швидкість відгуку інтерфейсу, але це поза межами нашої роботи, хоча може бути у нагоді при вдосконаленні нашої програми.

Таким чином, при використанні XCTest і OpenCV використовуючи Python, XCTest мав би виступати в якості основного агента для автоматизації взаємодії з користувачем і вимірювання метрик продуктивності, в той час як OpenCV служить додатковим агентом для збору і аналізу візуальних даних. Але було прийнято рішення більшу відповідальність надати саме модулю на Python, бо саме він відповідає за порівняння зображень, що і є основним етапом нашого

тестування. Цей комбінований підхід надає цінну інформацію про адаптивність інтерфейсів додатків для iOS, а саме які елементи не відповідають дійсності, в нашому випадку очікуваному результату наданим дизайнером, допомагаючи нам визначити області для оптимізації та покращення.

Хоч ці агенти працюють як окремі компоненти, але вони все одно гармонійно співпрацюють, використовуючи свої сильні сторони, створюють тандем, який розкриває всебічне розуміння адаптивності інтерфейсу. XCTest і OpenCV працюють в парі, об'єднуючи свої оцінки, щоб запропонувати всебічну оцінку за допомогою визначення некоректного інтерфейсу. Швидкі дії XCTest доповнюють візуальні спостереження OpenCV, забезпечуючи багатовимірну оцінку, яка охоплює як показники продуктивності, так і візуальні підказки(рис. 1.2). Разом ці агенти допомагають визначити області для покращення, такі місця де дизайн не відповідає дійсності та має бути виправлений у точних місцях по вказівках системи.

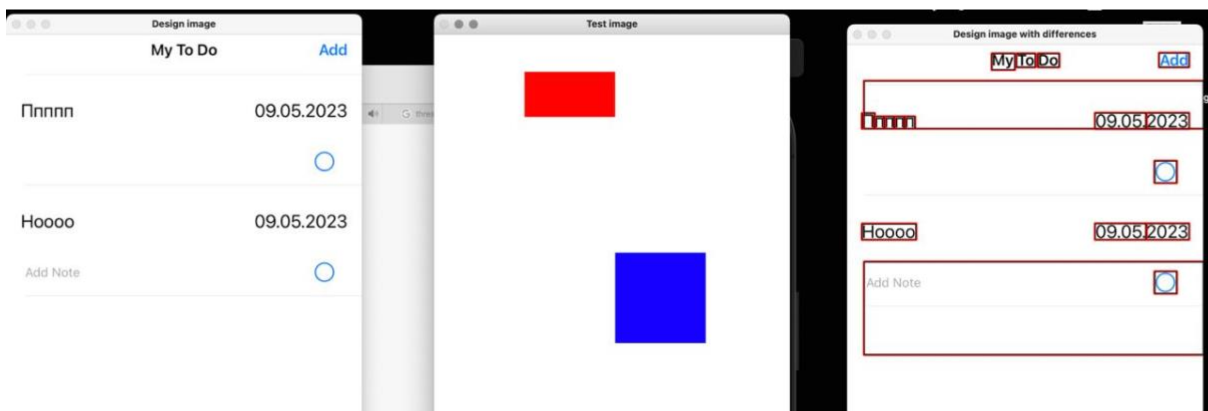

Рис. 1.2. Візуальні підказки при роботі тестовим порівнянням скріншотів

1.6 Висновки до розділу 1

Протягом цього розділу було розглянуто процес тестування на якому базується дана робота, зокрема тестування інтерфейсу. Показаний головний принцип тестування взагалі та розглянуті цілі тестування, їх головна суть.

Присутній також опис двох основних фреймворків XCTest та OpenCV, їхні особливості та напрями застосування безпосередньо у тестуванні. Розглянуто базовий клас фреймворку XCTestCase, його особливості та способи застосування. Він допомагає розробляти у своїй системі комплексні тестові кейси, виконувати твердження, обробляти асинхронні операції та вимірювати продуктивність у систематичний та організований спосіб Остання частина включає в себе співставлення двох фреймворків та загальний принцип взаємодії між ними.

РОЗДІЛ 2. Обґрунтування підходу до вирішення проблеми автоматизованого тестування адаптивності інтерфейсу на пристроях iOS/iPadOS

Автоматизоване тестування адаптивності інтерфейсу мобільних додатків стає ключовим фактором у забезпеченні якісного користувацького досвіду. Це дозволяє розробникам перевіряти реакцію додатків на різні розміри екрана, орієнтацію, роздільну здатність та інші параметри, які можуть впливати на відображення та функціональність інтерфейсу. Використання автоматизованих тестів допомагає виявляти значні помилки, забезпечує стабільну роботу додатків на різних пристроях і покращує загальний досвід користувача.

2.1 Підхід обраний для реалізації автоматизованої системи

Підхід, який ми застосовуємо для реалізації системи, використовує комбінацію OpenCV і XCTests для тестування додатків для iOS і iPadOS. Цей метод передбачає порівняння скріншотів розробленого користувацького інтерфейсу від дизайнера зі знімками, зробленими під час тестування, і позначення елементів, які не пройшли тест.

Щодо фреймворку XCTests, саме його можливості використовуються для збору матеріалів для тестування, OpenCV в свою чергу займаються обробкою та аналізом для порівняння скріншотів. Основними перевагами такого підходу є наступні пункти:

- автоматизоване тестування, бо якщо залучати ручне тестування, то воно є більш трудомістким, тривалим і схильним до помилок. Також, автоматизоване тестування можна запускати в будь-який час, стільки разів, скільки потрібно, і воно може виявити помилки, які можуть бути пропущені під час ручного тестування.

- візуальна перевірка, яка є основною частиною системи, бо порівнюючи скріншоти, ми проводимо візуальну перевірку програми, яка може виявити такі проблеми, як неправильне розташування елементів, неправильні кольори або неправильні розміри, які можуть бути пропущені при функціональному тестуванні.
- масштабованість є у цьому списку, бо після завершення початкового налаштування можливе додавання нових тестів або модифікація існуючих, щоб пристосувати їх до змін у додатку, а цей процес у свою чергу стає простішим і менш трудомістким.
- послідовність і надійність, під цим мається на увазі те, що автоматизовані тести можуть виконуватися послідовно і надавати надійні результати. Вони також дозволяють проводити регресійне тестування, щоб переконатися, що нещодавно внесені зміни або функції не порушили існуючу функціональність.

Хоч цей підхід має сильні переваги, він не позбавлений потенційних недоліків, про них нижче:

- по-перше, це початкові витрати часу, бо налаштування автоматизованого тестування, особливо з візуальним порівнянням, вимагає значних початкових витрат часу, які зумовлені бажанням отримати точний результат, бо візуальні елементи завжди є основою взаємодії користувача та додатків.
- по-друге, обслуговування, зміни в дизайні інтерфейсу часто можуть порушити роботу тестів, що вимагає постійного обслуговування набору тестів чи то додавання нових, чи то зміни старих.
- по-третє, негативні результати, які зазвичай з’являються не передбачувано, бо залежно від алгоритмів, що використовуються для

порівняння зображень, можуть бути випадки помилкових спрацьовувань (тест не спрацьовує, коли він повинен або навпаки) або помилкових негативних результатів (тест проходить, коли він не повинен).

- останній аспект це продуктивність, так як обробка зображень може вимагати значних обчислень, що може уповільнити тестування, особливо для великих додатків.

Зважаючи на всі переваги та недоліки можна сказати, що переваги автоматизованого тестування з візуальною верифікацією все одно можуть переважати недоліки, особливо для великих проектів з частим оновлюванням інтерфейсу користувача. Однак важливо зазначити, що така система має бути частиною ширшої стратегії тестування(рис. 2.1), що включає модульні тести, інтеграційні тести та ручне дослідницьке тестування, щоб забезпечити повне покриття.

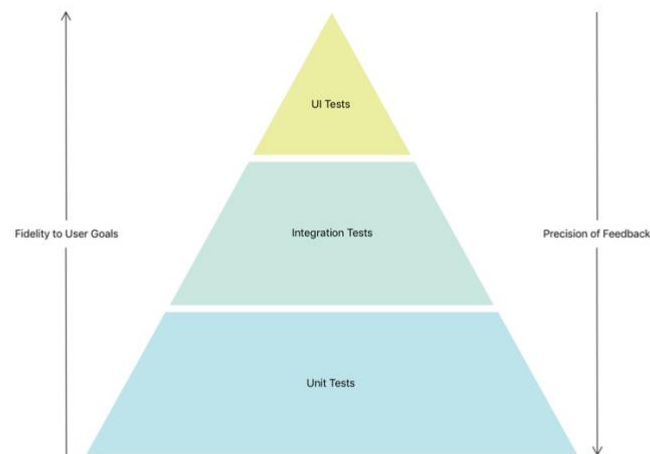

Рис. 2.1. Градація важливості тестів

2.2 Обґрунтування способу реалізації

Створення власної системи автоматизованого тестування адаптивності інтерфейсу може бути корисним, але й є складним завданням. Підхід з

використанням OpenCV та XCTest для автоматизованого тестування додатків для iOS/iPadOS є комплексним рішенням. Він спрямований на забезпечення узгодженості дизайну та зручність використання інтерфейсів додатків шляхом порівняння скріншотів дизайну з тестовими скріншотами. Цей підхід є дуже практичним, оскільки він зменшує ручну роботу, необхідну на етапі тестування, підвищує точність, усуваючи людські помилки, і прискорює процес тестування. Обраний підхід також дозволяє виявити будь-які зміни або аномалії в користувацькому інтерфейсі (UI), які можуть бути пропущені при ручному тестуванні. Це можуть бути зміни в розташуванні елементів, невідповідність кольорів, відсутні або зайві елементи тощо. Саме такі раніше перераховані неточності має виявляти розроблювана система.

OpenCV (Open Source Computer Vision) - бібліотека програмних функцій, призначених для комп'ютерного зору в реальному часі. З її допомогою можна аналізувати скріншоти, ідентифікувати елементи інтерфейсу та порівнювати їхні властивості (положення, колір, розмір тощо) з проектними скріншотами, саме ці інструменти і використані при вирішенні нашої проблеми. XCTest - це фреймворк для тестування інтерфейсу користувача для iOS додатків від Apple. Він дозволяє тестувальникам писати UI тести, які відображають взаємодію користувачів з додатком, а саме допомагає створювати заплановані події, які автоматично обчислюються як правильні та неправильні, якщо якась умова не виконалась. XCTest використовується у нашій системі для створення тестових скріншотів під час тестування взаємодії.

Цей підхід перевершує інші насамперед завдяки високому рівню автоматизації та точності, бо навіть найменші неточності будуть вважатися за помилки. Хоча ручне тестування може бути суб'єктивним і схильним до людських помилок, цей підхід забезпечує об'єктивність і повторюваність. Він робить процес тестування швидшим, більш ефективним і послідовним, скорочуючи час виходу на ринок. Цей підхід також масштабується і може бути

легко інтегрований в конвеєр CI/CD, що робить його частиною звичайного процесу збірки.

Ми реалізуємо тип візуального регресійного тестування, де ми робите скріншоти нашого додатку, а потім порівнюєте їх з еталонним дизайном. Наш метод, який використовує XCTest для створення скріншотів і OpenCV в скрипті Python для порівняння, можемо виправдати саме такими конкретними чинниками:

- Автоматизація ручної роботи:
    - Традиційно тестування графічного інтерфейсу програми може включати багато ручної роботи, переглядаючи кожен екран, щоб переконатися, що він відповідає дизайну. Автоматизувавши цей процес, ми, з більшою вірогідністю, заощадите багато часу і зменшимо кількість людських помилок.
- Послідовність:
    - XCTest дозволяє робити послідовні скріншоти при кожному запуску тестів, гарантуючи, що одні й ті ж візуальні елементи завжди перевіряються однаково.
- Точність:
    - OpenCV - це надійна бібліотека для задач комп'ютерного зору та обробки зображень. Використовуючи такі функції, як cvtColor, поріг і пошук контурів, ми можемо точно визначити відмінності між проектним і тестовим зображеннями аж до рівня пікселів.
- Швидкість:
    - Автоматизовані тести можна запускати в будь-який час, і вони зазвичай виконуються швидше, ніж ручне тестування. Вони можуть бути включені як частина нашого конвеєра безперервної інтеграції/доставки, швидко надаючи розробникам зворотній зв'язок про візуальну узгодженість їхніх змін.

- Об'єктивність:
  - Автоматизоване тестування усуває суб'єктивні судження з процесу. Дві людини можуть не погодитися з тим, чи достатньо близький дизайн до запланованого вигляду, але алгоритм буде послідовно застосовувати однаковий рівень строгості.
- Масштабованість:
  - У міру того, як наш додаток зростає і включає більше екранів або візуальних елементів, час, необхідний для ручного тестування, також буде збільшуватися. Однак наш метод зможе впоратися з більшими додатками так само легко.
- Можливість багаторазового використання:
  - Тести, написані для однієї частини програми, часто можна повторно використовувати для інших, заощаджуючи зусилля при створенні тестів. Це також забезпечує однаковість тестування різних частин програми.
- Документація:
  - Тести служать формою документації, що описує очікувану поведінку програми. Це може бути корисно для нових членів команди або для майбутніх користувачів, щоб зрозуміти, що очікується від вигляду та роботи додатку.

Однак треба мати на увазі, що така форма тестування не може замінити всі форми ручного тестування. Люди-тестери краще здатні оцінити суб'єктивні аспекти дизайну, такі як естетика або зручність використання. Крім того, автоматизовані тести можуть знайти тільки ті типи помилок, для яких вони спеціально написані. Вони навряд чи виявлять несподівані або нові проблеми. Отже, хоча наш метод XCTest і OpenCV має багато переваг, його слід було б використовувати в поєднанні з іншими методами тестування.

2.3 Обґрунтування вибору фреймворку OpenCV для тестування інтерфейсу

Для порівняння файлів зі скріншотами було обрано універсальну бібліотеку, яка відома своїми ефективними можливостями комп'ютерного зору в реальному часі, вона має кілька ключових переваг, що роблять її підходящим вибором для тестування інтерфейсів.

- універсальність та функціональність у OpenCV постачається з широким спектром алгоритмів обробки зображень та відео. Ці алгоритми можна використовувати для виконання таких операцій, як виявлення та розпізнавання облич, ідентифікація об'єктів, класифікація дій людини на відео, відстеження руху камери тощо. В контексті тестування інтерфейсу OpenCV ми будемо використовувати для ідентифікації елементів інтерфейсу на скріншотах і порівняння їх властивостей з проектними скріншотами.
- простота використання з Python, бо якщо трохи заглибитись, то можна дізнатись, що OpenCV-Python - це Python-обгортка для оригінальної реалізації OpenCV C++. Python - це мова програмування високого рівня, відома своєю читабельністю і простотою вивчення, що робить реалізацію складних алгоритмів швидшою і простішою. Простота Python у поєднанні з потужними можливостями OpenCV є вагомим аргументом на користь використання OpenCV-Python у нашому фреймворку для тестування інтерфейсу.
- продуктивність, бо OpenCV відомий своєю високою продуктивністю. Він розроблений для обчислювальної ефективності з сильним акцентом на додатки, що працюють у реальному часі. Це гарантує, що навіть з великомасштабними або складними зображеннями тестовий фреймворк може працювати швидко і ефективно.

- відкритий вихідний код і підтримка спільноти - OpenCV має відкритий вихідний код, що означає, що він безкоштовний у використанні і має велику спільноту розробників, які роблять свій внесок у його розвиток і допомагають з усуненням існуючих несправностей. Це робить його надійним вибором для проекту будь-якого масштабу, тому для нашого він стане доречним доповненням.
- інтеграція - OpenCV можна легко інтегрувати з іншими бібліотеками та фреймворками. Наприклад, якщо у нас є проект на Python і ми використовуємо бібліотеку машинного навчання, таку як TensorFlow або PyTorch, ми можете легко поєднати OpenCV з цими бібліотеками для обробки зображень перед подачею їх у модель машинного навчання. OpenCV має функції для завантаження, обробки та збереження зображень, а також для виконання розпізнавання обличчя, виявлення об'єктів та інших завдань комп'ютерного зору.
- незалежність від платформи, бо OpenCV підтримує широкий спектр операційних систем, включаючи Windows, Linux і MacOS, що дозволяє створити більш гнучке і універсальне середовище тестування. Не дуже дійсна перевага у нашому випадку, але якщо дивитися на розширення спектру тесту пристроїв, то ця перевага дуже стане у нагоді.

Використовуючи можливості OpenCV, ми можемо автоматизувати процес порівняння скріншотів і пошуку розбіжностей, підвищуючи швидкість, надійність і ефективність процесу тестування, тому був обраний саме цей варіант для створення системи.

2.4 Порівняння системи з вже існуючими

Створення автоматизованої системи вимагає складного процесу проектування та розробки самої системи, тому підходи до виконання у всіх

різні. Прикладом для порівняння є open-source систему SnapshotTesting вона перевіряє скріншоти за допомогою функції assrestSnapshot(). У цій системи порівнюються скріншоти прив'язані до конкретних view. Слід зауважити, що ця система більш досконала, якщо порівнювати глобальний результат. Але наша система має перспективи на більш зручну автоматизовану систему. Планується реалізувати єдину команду, яка буде повністю злагоджено увесь процес задуманого тестування, створення скріншотів та порівняння їх за допомогою засобів обробки зображень.

Враховуючи виклики, що виникають при розробці автоматизованої системи, ми розробляємо підхід, який спрямований на полегшення та оптимізацію процесу. Наша система, в ідеалі, буде включати механізми для автоматичного створення, зберігання та порівняння скріншотів. Це допоможе значно зменшити час, який потрібен для тестування, а також зменшити можливість помилок, що виникають при ручному виконанні цих завдань.

У контексті тестування, важливим є не тільки абсолютна точність, але й здатність легко та ефективно здійснювати порівняння скріншотів. Ми плануємо використати передові технології обробки зображень за допомогою інструментів OpenCV-Python для створення нашої системи, що дозволить нам виконувати глибоке порівняння зображень, не обмежуючись лише поверхневим аналізом. Це може включати в себе порівняння контрасту, колірної гами, і навіть індивідуальних пікселів на скріншотах, загалом можна реалізувати різні способи порівняння та обробки скріншотів.

Окрім того, ми прагнемо розробляти систему так, щоб вона була максимально гнучкою та налаштовуваною. Використовуючи вбудовані параметри, користувачі зможуть налаштовувати процес тестування згідно своїх потреб, включаючи вибір конкретних екранів додатку для створення скріншотів за допомогою засобів XCode або наші особисті підходи та встановлення порогу схожості для порівняння зображень. Це дозволить

системі адаптуватися до широкого спектра сценаріїв тестування, забезпечуючи більш ефективний процес тестування.

2.5 Висновки до розділу 2

У цьому розділі було проведено обґрунтування підходу по вирішенню даної проблеми створення автоматизованого тестування адаптивності інтерфейсу мобільних додатків на базі iOS та пояснення чому був обраний варіант реалізації у поєднанні XCTest та OpenCV. Також проведено порівняння перспектив та поточний стан досліджуваних програм.

РОЗДІЛ 3. ОПИС ІМПЛЕМЕНТАЦІЇ ТА АНАЛІЗ РЕЗУЛЬТАТІВ

3.1 Опис імплементації системи порівняння скріншотів

Суть порівняння фотографій полягає в накладанні фільтрів на них для створення однорідної картинки, яка буде придатна до порівняння. Самі фотографії приходять у зазначену директорію після встановленого нами порядку тестування у XCode за допомогою інструментів вище згаданого фреймворку XCTest. Після надходження скріншотів до вказаних папок у нас вже є можливість порівняти скріншоти з тестів до варіантів від дизайнера. Важливо зауважити, що скріншоти мають відповідати певним ім’ям для того, щоб було зрозуміло які пари мають тестуватися та для точності співставлення скріншотів з дизайну та тестів. Після запуску Python скрипту виконується сам процес тестування, відбувається прохід по двом директоріям DesignScreenshots та TestScreenshots(рис. 3.1), які у свою чергу мають піддиректорії з розмірами екранів доступних на пристроях iOS від 4,7 до 6,7 дюймів, а саме екрани, які мають поточну підтримку оновлень iOS від Apple. Поділ підгруп відбувається в залежності від висоти екрану пристрою, бо саме таку висоту має скріншот зроблений за допомогою тестів та скріншот, який надається дизайнером. Система доходить по директоріям до файлів фото та за порядком порівнює скріншоти, саме тому надважливо створювати ім’я по шаблону з назви дизайн-скріншоту. За допомогою засобів OpenCV скріншоти порівнюються й у разі несумісності якихось елементів створюється колаж, який містить у собі по порядку дизайн-скріншот, тест-скріншот та скріншот на якому позначено елементи які на скріншоті тестовому не співпадають з варіантом з дизайну. Для таких колажів створюється окрема папка FailedTestScreenshots, яка зберігає всі варіанти, які не пройшли тест та мають несумісності.

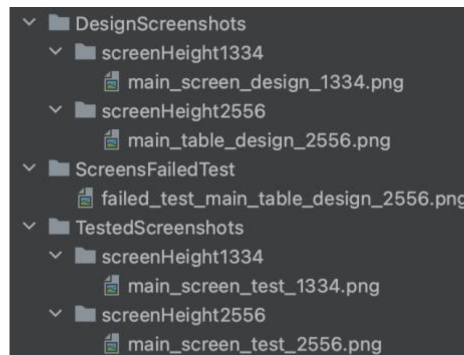

Рис. 3.1 Ієрархія папок з специфічними розмірами екранів

3.2 Опис засобів застосованих для імплементації системи(XCTest)

Як вже було зазначено вище у роботі використовується два фреймворки – це XCTest та OpenCV-Python. Розпочнемо з засобів XCTest, для збору скріншотів потрібне лиш правильний стан програми на екрані та своєчасне створення скріншоту за допомогою команди: XCUIScreen.main.screenshot().

Зберігати скріншот потрібно у деяку змінну, щоб потім можна було дістати звідти фото, яке будемо вантажити у директорію для тестування.
Приклад створення та збереження скріншоту(рис. 3.2) показаний нижче:

```
func testCaptureScreenshots() throws {
    let app = XCUIApplication()
    app.launch()
    let testedScreenshotsDir =
        "/Users/dmytrohetman/PycharmProjects/testCounterProject/TestedScreenshots"

    let mainScreenScreenshot = XCUIScreen.main.screenshot()
    guard let mainScreenHeight = mainScreenScreenshot.image.cgImage?.height else { return }
    saveImageToFile(image: mainScreenScreenshot.image, imageName:
        "main_screen_test_\(mainScreenHeight)", directory:
        "\(testedScreenshotsDir)/screenHeight\(mainScreenHeight)")
}
```

Рис. 3.2. Створення та запис скріншоту за допомогою XCTest та Swift

Під час збереження файлу у потрібну директорію застосовується окремий метод saveImageToFile(…)(рис. 3.3), який і відповідає за точне збереження файлу у

файловій системі комп'ютера. Головний нюанс треба використовувати для збереження саме формат PNG, бо за методикою стиснення без втрат, воно не змінює базові дані матриці під час стиснення. Якщо використовувати JPG, то техніка стиснення з втратами може замінювати базові дані деякими найближчими округленими значеннями для економії місця, що буде означати неточності у нашому тестуванні.

```
private extension TODOListUITests {
    func saveImageToFile(image: UIImage, imageName: String, fileExtension: String = "png", directory: String) {
        guard let data = (fileExtension == "png") ? image.pngData() : image.jpegData(compressionQuality: 1.0) else {
            print("Failed to convert image to data.")
            return
        }
        let fileManager = FileManager.default
        let projectDirectoryURL = URL(fileURLWithPath: directory, isDirectory: true)
        do {
            try fileManager.createDirectory(at: projectDirectoryURL, withIntermediateDirectories: true, attributes: nil)
            let imageURL = projectDirectoryURL.appendingPathComponent("\(imageName).\(fileExtension)")
            try data.write(to: imageURL)
            print("Image saved at: \(imageURL.path)")
        } catch {
            print("Error saving image: \(error)")
        }
    }
}
```

Рис. 3.3. Функція для збереження скріншоту у вказану директорію

Тестування методу можна починати різними способами, розглянемо їх, але на практиці застосовувався лише підхід запуску напряму з XCode. Отже, існує кілька способів запуску кейсів XCTest.

- Через інтерфейс Xcode:

  Це найпоширеніший спосіб, який був використаний як зазначено раніше. У Xcode ми можемо запустити тести за допомогою пункту меню "Product -> Test" або іконок у вигляді ромбів у навігаторі тестів. Ми також можемо натиснути комбінацію клавіш Command+U, щоб запустити всі тести.

- Командний рядок:

  Тести можна запустити з терміналу за допомогою команди xcodebuild test з наступними необхідними параметрами. Це корисний метод для запуску тестів в рамках конвеєра безперервної інтеграції.

- Test Plan:

  Xcode 11 і вище надає опцію під назвою " Test Plans". Плани тестів дозволяють запускати тести з різними конфігураціями і керувати складністю тестування в різних середовищах.

- Конфігурація схеми:

  Ми можемо змінити схему, щоб включити або виключити певні класи тестів або тестові кейси. За замовчуванням, дія Test у схемі запускає всі тести. Але ми можемо вказати, щоб вона виконувала лише деякі тести, нприклад, тільки testScreenshotsCapturing, або вказати порядок запуску тестів.

- Інструменти CI/CD:

  Інструменти безперервної інтеграції, такі як Jenkins, CircleCI, Travis CI тощо, можна використовувати для запуску наборів XCTest як частину автоматизованих конвеєрів збірки і тестування.

- Симулятор або реальний пристрій:

  Тести можна запускати як на симуляторі, так і на реальному пристрої в залежності від вимог.

- Паралельне тестування:

  Xcode 10 і вище надає можливість запускати тести паралельно на декількох симуляторах. Це може значно скоротити загальний час, необхідний для виконання всіх тестів.

Кожен метод має свої плюси і мінуси, тому вибір найкращого методу буде залежати від наших конкретних подальшиз потреб на певному проекті.

3.3 Опис засобів застосованих для імплементації системи(OpenCV)

Було прийняте рішення використовувати засоби фреймворку OpenCV-Python. Спочатку ми обрізаємо у скріншота усіх iPhone status bar, бо різні моделі мають різні показники заряду, зв’язку або назву оператору, тому важко передбачити це на всіх тестових пристроях, а в свою чергу різні дані будуть провокувати непройдений тест, що по суті є хибною поведінкою. Після обрізання скріншоту маємо готовий зразок для тестування.

Наступним етапом буде конвертація у відтінки сірого за допомогою функції OpenCV cvtColor() з прапором COLOR_BGR2GRAY(рис. 3.4). Це робиться тому, що зображення у відтінках сірого легше аналізувати та обробляти, ніж кольорові, особливо коли нас цікавлять зміни структури та інтенсивності, а не зміни кольору. Метод cv2.cvtColor() використовується для перетворення зображення з одного колірного простору в інший. У OpenCV існує понад 150 методів перетворення колірних просторів. Під капотом cv2.cvtColor() використовує декілька методів та алгоритмів для виконання цього перетворення колірного простору. Перший з них представлення колірного простору, сам OpenCV представляє зображення як масив NumPy з трьома колірними каналами (червоний, зелений і синій) або чотирма каналами (червоний, зелений, синій і альфа). Перетворення колірного простору виконується на основі значень цих каналів. Кожен колірний простір має власну матрицю перетворення, яка визначає співвідношення між колірними каналами. Наприклад, перетворення з колірного простору RGB у відтінки сірого можна здійснити за допомогою формули:

$$Y = 0{,}299 * R + 0{,}587 * G + 0{,}114 * B \,,$$

де Y – результуюче значення інтенсивності у відтінках сірого, а R, G і B – значення червоного, зеленого і синього каналів вихідного зображення відповідно.

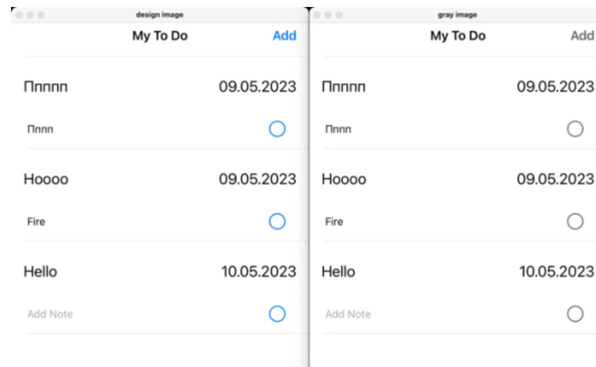

Рис. 3.4. Колір фото без та з конвертацією COLOR_BGR2GRAY

Далі визначається індекс структурної подібності (SSIM). SSIM – це метод порівняння подібності між двома зображеннями. Оцінка SSIM і зображення різниці обчислюються між двома зображеннями у відтінках сірого за допомогою функції structural_similarity(). Оцінка потім роздруковується у відсотках. Зображення різниці масштабується до діапазону 0-255 і перетворюється у 8-бітне ціле число без знаку для подальшої обробки. Функція structural_similarity() не є частиною OpenCV-Python, а є частиною бібліотеки scikit-image (skimage.metrics). Він особливо корисний для оцінки якості алгоритмів обробки зображень, таких як стиснення, згладжування та покращення.

Розрахунок SSIM включає три компоненти:

- порівняння яскравості (L): Цей компонент вимірює схожість середньої яскравості двох зображень.
- порівняння контрасту (C): Цей компонент вимірює схожість контрасту (різницю між найяскравішими і найтемнішими ділянками) двох зображень.
- структурне порівняння (S): Цей компонент вимірює схожість локальних піксельних візерунків або текстур.

Значення SSIM обчислюється шляхом перемноження цих трьох компонентів:

$$SSIM(x, y) = L(x, y) * C(x, y) * S(x, y)$$

Індекс SSIM знаходиться в діапазоні від -1 до 1, причому значення 1 означає, що зображення ідентичні, а значення, близькі до 0, вказують на низьку схожість.

Наступний крок це бінарна маска проектного зображення, вона створюється шляхом застосування операції порогової обробки з бінаризацією Оцу. Пікселі у відтінках сірого, значення яких менше певного порогового значення, стають чорними (0), а ті, значення яких більше порогового значення, стають білими (255). Таке бінарне зображення використовується для виділення потрібних областей на проектному зображенні.

Основною функцією на цьому етапі є cv2.threshold. Ця функція використовується для застосування порогової операції до зображення. Вона приймає чотири параметри: вихідне зображення, порогове значення, максимальне значення, яке буде задано, якщо значення пікселя перевищує порогове значення, і тип порогового значення, яке буде застосовано. Cv2.THRESH_BINARY_INV або cv2.THRESH_OTSU: Це тип порогового значення, яке буде застосовано. cv2.THRESH_BINARY_INV означає, що якщо інтенсивність пікселя більша за встановлений поріг, значення буде встановлено на 0, інакше буде встановлено максимальне значення. Це обернений до cv2.THRESH_BINARY. cv2.THRESH_OTSU використовує метод бінаризації Оцу. Цей метод обчислює «оптимальний» поріг (значення від 0 до 255), який мінімізує зважену внутрішньокласову дисперсію. Кінцевим результатом рядка, який має попередні елементи буде бінарне зображення design_image_mask з інвертованою інтенсивністю градацій сірого (тобто світліші пікселі стають темнішими і навпаки), а поріг для визначення темного/світлого обчислюється автоматично за методом Оцу.

Основною та найбільш явною частиною обробки зображення це є знаходження контуру(рис. 3.5). findContours() – функція в бібліотеці OpenCV для Python, яка використовується для виявлення та малювання контурів у бінарних зображеннях. Ця функція часто використовується в таких задачах, як виявлення

об'єктів, аналіз форми та розпізнавання об'єктів. До різницевого зображення застосовується ще одна операція порогування, і за допомогою функції findContours() на пороговому різницевому зображенні знаходять контури (неперервні криві, що з'єднують усі неперервні точки вздовж границі). Нарешті, findContours() повертає контури та ієрархію. Кожен окремий контур – це список координат (x, y) граничних точок об'єкта у форматі Python. Спосіб повернення цих координат можна контролювати за допомогою другого аргументу – методу апроксимації контуру. Для кожного знайденого контуру, якщо його площа більша за 40, перевіряється, чи знаходиться він у межах маски розрахункового зображення. Якщо так, то навколо контуру на растрі малюється прямокутник. Це позначає область, де проектне зображення відрізняється від тестового.

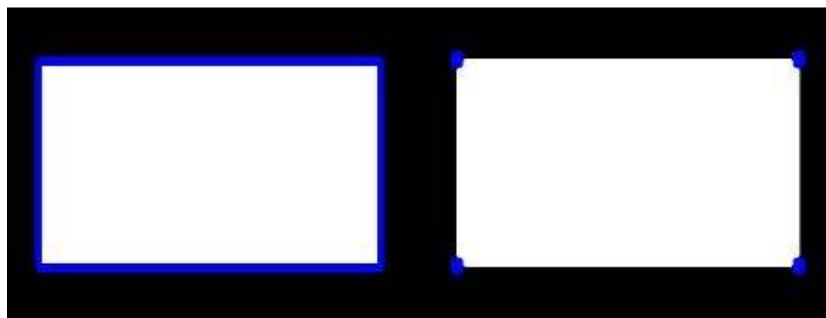

Рис. 3.5. Приклад визначення контурів за допомогою findContours()

3.4 Аналіз результатів

Розробка забезпечення для тестування з автоматизованим підходом займає більше часу через більший спектр можливих варіантів роботи системи та підтримка цих всіх методів. Виконана робота показує можливості інтеграції Python програми у iOS тестування, хоча воно працює не на пряму, але можливості поєднання існують, можливе покращення це поєднання скриптів запуску тестів через MacOS console applications та запуску повного циклу програми за допомогою єдиної команди. Загалом система визначає проблемні місця, але все одно потребує вдосконалення як в плані оптимізації, так і в плані точності визначення проблемних місць інтерфейсу певного додатку. Планується подальше

дослідження використаних та можливих нових підходів сканування скріншотів для визначення проблемних місць та елементів, які явно можна визначити як помилкові та мають бути виправлені.

У результаті у нас є автоматизована система тестування, де ми використовуємо XCTest для створення скріншотів нашого додатку, а потім за допомогою скрипта на Python з OpenCV порівнюємо дизайн і тестові зображення, щоб знайти відмінності.

Це дуже ефективний спосіб проведення візуальних регресійних тестів - тестів, які перевіряють, що користувальницький інтерфейс виглядає так, як очікує кінцевий користувач. Нижче наведений аналіз цього підходу.

Щодо застосування такого підходу тестування можна визначити наступні переваги. По-перше, те, що воно автоматизоване. Це може значно скоротити час і ресурси, необхідні для проведення тестів, оскільки їх можна запускати автоматично.

По-друге, у нашому підході використовується візуальна перевірка. Порівнюючи скріншоти, ми можемо виявити візуальні розбіжності, які можуть бути пропущені при інших формах тестування.

По-третє, виходить, що на підхід застосовує широке охоплення. Ми можемо охопити багато сценаріїв і станів програми, а не тільки функціональну коректність. Незалежність від вихідного коду, тобто наші модулі з XCTest та OpenCV системами виходять незалежними. Цей вид тестування не залежить від вихідного коду, тобто його можна використовувати в різних версіях або навіть абсолютно різних додатках.

Щодо недоліків, то виокремимо наступні зауваження. По-перше, це чутливість до незначних змін. Тести можуть не пройти через незначні зміни в інтерфейсі, які не обов'язково є помилками.

По-друге, має бути високий рівень обслуговування, тобто скріншоти потрібно регулярно оновлювати, щоб відображати останній стан програми,

особливо на активних фазах розробки. Це може вимагати значного часу і зусиль, що буде уповільнювати процес тестування та розробки загалом.

По-третє, присутнє обмежене розуміння, що під цим мається на увазі, це те, що скріншоти не дають розуміння того, чому тест не пройшов. Вони можуть сказати нам, що щось не так, але не те, що саме не так.

Давайте розглянемо, які існують напрями ідей щодо покращення нашої системи. По-перше, можна застосувати перцептивні відмінності. Ми можемо розглянути інструменти перцептивного розрізнення, які призначені для ігнорування незначних відмінностей, таких як незначні зміни в кольорі або положенні пікселів.

По-друге, додавання порогу толерантності. Введення порогу толерантності може допомогти ігнорувати незначні зміни, які знаходяться в межах прийнятного діапазону.

По-третє, інтеграція з CI/CD. Інтеграція цього процесу з нашим конвеєром безперервної інтеграції/безперервної доставки може зробити цикл зворотного зв'язку щодо змін набагато швидшим, що забезпечить швидші ітерації та кращу якість.

3.5 Висновки до розділу 3

Цей розділ наповнений відомостями про імплементацію автоматизованої системи тестування адаптивності інтерфейсу iOS додатків за допомогою потужностей OpenCV-Python фреймворку та XCTest, описані особливості роботи цих систем та нюанси роботи засобів під капотом. Проведено аналіз результатів виконаної роботи та зазначено моменти, які слід покращити протягом подальшої підтримки продукту. Визначені переваги та недоліки виконаної системи, дано короткий опис кожного пункту.

ВИСНОВКИ

Отже, можна зробити висновки з результатів виконаної роботи. Ми виконали дослідження фреймворків XCTest та OpenCV-Python загально та в контексті розробленої автоматизованої системи тестування адаптивності інтерфейсу iOS додатків. Було розглянуто переваги та недоліки системи та засобів застосованих для розробки. Здійснено порівняння з існуючим фреймворком SnapshotTesting. Досліджені слабкі та сильні місця програми та поточного підходу. Глибоке дослідження інструментів застосованих у роботі дозволяє зрозуміти принцип роботи порівняння зображень, та знову ж таки, побачити переваги та недоліки обраного підходу.

Тестування системою проводилося на додатках створених протягом навчання, у цілому результат задовільний, але присутні певні неточності та недосконалі результати, які мають бути виправлені. Загалом система розрізняє відмінності тексту, прямокутних елементів на екрані, неточності спостерігаються на елементах світло-сірого кольору. Але слід зазначити, що більшість елементів система визначає правильно, отже, поставлених цілей ми досягли. Зручність поєднання двох систем теж дає свої плоди, воно покращує роботу з розробкою системи та роботою з нею загалом.

Щоб покращити систему тестування треба спробувати виправити існуючий підхід. Незважаючи на це, була створена система, яка має великі перспективи на майбутнє в плані розширення функціоналу та інструментарію, більша оптимізація та зручність використання. Гарним вдосконаленням буде з’єднання запуску XCTest та Python програми послідовно. Це буде більш зручний підхід саме для тестувальників, бо так виходить система напівавтоматизована. Також варто зауважити, що система може бути вдосконалена додаванням можливості тестування iPad екрани

Загалом система виявилася гарним прикладом для фундаменту системи тестування, який можна визначити як Proof of Concept. Досліджуваний метод

вийшов гарним плацдармом для подальшого розвитку даної автоматизованої системи тестування адаптивності інтерфейсу iOS/iPadOS додатків.

ДЖЕРЕЛА

1. XCTest | Apple Developer Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://developer.apple.com/documentation/xctest.
2. OpenCV: Thresholding [Електронний ресурс] – Режим доступу до ресурсу: https://docs.opencv.org/4.x/d7/d4d/tutorial_py_thresholding.html.
3. OpenCV: Contours [Електронний ресурс] – Режим доступу до ресурсу: https://docs.opencv.org/4.x/d4/d73/tutorial_py_contours_begin.html.
4. Image Difference with OpenCV and Python – PyImageSearch [Електронний ресурс] – Режим доступу до ресурсу: https://pyimagesearch.com/2017/06/19/image-difference-with-opencv-and-python/.
5. Mobile App Development Lifecycle [Електронний ресурс] – Режим доступу до ресурсу: https://www.itrobes.com/mobile-app-development-lifecycle/.
6. OpenCV: Color Space Conversions [Електронний ресурс] – Режим доступу до ресурсу: https://docs.opencv.org/3.4/d8/d01/group__imgproc__color__conversions.html.
7. Module: metrics — skimage v0.20.0 docs [Електронний ресурс] – Режим доступу до ресурсу: https://scikit-image.org/docs/stable/api/skimage.metrics.html.
8. Testing your apps in Xcode | Apple Developer Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://developer.apple.com/documentation/xcode/testing-your-apps-in-xcode.
9. XCTestCase | Apple Developer Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://developer.apple.com/documentation/xctest/xctestcase.