МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)

Кафедра Програмної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

рівень вищої освіти другий (магістерський)

Дослідження методів ефективної комунікацій між UI дизайнерами та розробниками
(тема)

Виконав:
Випускник II курсу, групи ІПЗм-19-3
Гусаков Є.В.
(прізвище, ініціали)

Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва спеціальності)

Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Керівник доцент Лановий О.Ф.
(посада, прізвище)

Допускається до захисту
Зав. кафедри _____ З.В. Дудар
(підпис) (прізвище, ініціали)

2021 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)

Кафедра Програмної інженерії
(повна назва)

Рівень вищої освіти другий (магістерський)

Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва)

Тип програми освітньо-наукова програма
(освітньо-професійна або освітньо-наукова)

Освітня програма Інженерія програмного забезпечення
(повна назва)

ЗАТВЕРДЖУЮ:

Зав. кафедри _____
(підпис)
« 26 » березня 2021 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студента Гусакову Єгору Володимировичу
(прізвище, ім'я, по батькові)

1. Тема роботи «Дослідження методів ефективної комунікацій між UI дизайнерами та розробниками»

затверджена наказом університету від 26.03.2021 № 385 Ст

2. Термін подання студентом роботи до екзаменаційної комісії «10» травня 2021 р.

3. Вихідні дані до роботи *методи комунікації в софтверних компаніях, розробка чат-ботів, електронні ресурси за темою роботи, вимоги до реалізації.*

4. Перелік питань, що потрібно опрацювати в роботі *мета роботи, аналіз предметної області та постановка задачі, методи порівняння зображень, особливості розробки чат-ботів, опис програмної реалізації, висновки по роботі.*

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, ілюстрацій (слайдів) *мета роботи, обґрунтування дослідження, постановка задачі, методи і алгоритми, структурно-логічна схема взаємодії даних, опис отриманих результатів, інтерфейс програмної системи, демонстраційні матеріали*

6 Консультанти розділів роботи

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата
Спецчастина	Лановий О.Ф.		

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Аналіз предметної галузі	25.01.21 – 08.02.21	*Виконано*
2	Аналіз аналогів	4.02.21 – 12.02.21	*Виконано*
3	Постановка задачі дослідження	09.02.21 – 16.02.21	*Виконано*
4	Дослідження методів порівняння зображень	15.02.21 – 28.02.21	*Виконано*
5	Планування досліджень	01.03.21–05.03.21	*Виконано*
6	Розробка структури ПЗ та програмна реалізація	05.03.21 – 25.03.21	*Виконано*
	Проведення експериментів та аналіз результатів	25.03.21 – 08.04.21	*Виконано*
7	Доопрацювання ПЗ	08.04.21 – 16.04.21	*Виконано*
	Підготовка статі за темою дослідження	15.04.21 – 24.04.21	*Виконано*
	Підготовка пояснювальної записки	18.04.21 – 08.05.21	*Виконано*
8	Нормоконтроль, рецензування	08.05.21 – 11.05.21	*Виконано*
9	Занесення диплома в електронний архів	12.05.21	*Виконано*
10	Допуск до захисту у зав. кафедри	13.05.21	*Виконано*

Дата видачі завдання ____ _____ 2021р.

Студент _____
(підпис)

Керівник роботи _____ доц. Лановий О.Ф.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ/ ABSTRACT

Кваліфікаційна робота магістра містить: 82 с., 25 рис., 28 джерел.

UI, КОМУНІКАЦІЇ, ДІЗАЙНЕР, РОЗРОБНИК, МЕСЕНДЖЕР, ЧАТ-БОТ

Об’єкт дослідження: методи організації комунікації між дизайнерами та розробниками в межах колективів розробників ПЗ.

Мета роботи: проведення досліджень для виявлення шляхів підвищення ефективності комунікаційної взаємодії між учасниками розробки ПЗ за рахунок використання додаткових інструментальних засобів інформаційної взаємодії – чат-ботів.

Результатом роботи є прототип програмної системи, орієнтованої на підвищення ефективності комунікаційної взаємодії між дизайнерами та розробниками, який інтегровано в корпоративний месенджер.

UI, COMMUNICATIONS, DESIGNER, DEVELOPER, MESSAGER, CHAT BOT

Object of research: methods of communication between designers and developers within the teams of software developers.

Purpose: to conduct research to identify ways to improve the effectiveness of communication between participants in software development through the use of additional tools of information interaction - chatbots.

The result will be a prototype software system focused on improving the efficiency of communication between designers and developers, selected means of software implementation of the software system, which will be integrated into the corporate messenger.

Я, *Гусаков Єгор Володимирович,* студент групи ІПЗм-19-3, здобувач вищої освіти на другому (магістерському) рівні кафедри «Програмна інженерія», заявляю: моя кваліфікаційна робота на тему «Дослідження методів ефективної комунікацій між UI дизайнерами та розробниками», що буде представлена в екзаменаційну комісію для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в електронному архіві відкритого доступу ElAr KhNURE. Всі запозичення з друкованих та електронних джерел мають відповідні посилання.

Я ознайомлений(а) з діючим положенням «Про протидію академічному плагіату в ХНУРЕ», згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування дисциплінарни

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ 8
ВСТУП 9
1 АНАЛІЗ ПРОБЛЕМНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ 11
1.1 Поняття комунікації 11
1.2 Роль дизайнерів UI в процесі розробки ПЗ 12
1.3 Аналіз існуючих рішень 16
1.4 Сучасні концепції побудови UI 18
1.5 Постановка задачі дослідження 22
2 РОЛЬ ДИЗАЙНЕРА UI В КОМАНДНІЙ РОЗРОБЦІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 24
2.1 Підходи до розробки ПЗ 24
2.2 Гнучка розробка програмного забезпечення 25
2.3 Роль та значення дизайнера UI в команді 29
3 ТЕХНОЛОГІЇ ЗАБЕЗПЕЧЕННЯ КОМУНІКАЦІЇ 34
3.1 Необхідність постійної комунікації 34
3.2 Програмний засіб для покращення якості комунікації 37
3.3 Алгоритми порівняння зображень 38
4 ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ 43
4.1 Архітектура чат-бота 43
4.2 Вибір засобів програмної реалізації 45
4.3 Програмна реалізація чат-бота та робота з додатком 48
ВИСНОВКИ 55
ПЕРЕЛІК ПОСИЛАНЬ 56
ДОДАТОК А.Перелік джерел посилання за науковими напрямами керівника та науковців кафедри програмної інженерії 60
ДОДАТОК Б. Звіт результатів перевірки кваліфікаційної роботи на унікальність тексту 61
ДОДАТОК В. Слайди презентації 62

ДОДАТОК Г. Фрагмент лістингу коду 73
ДОДАТОК Д. Апробація результатів роботи 75
ДОДАТОК Е. Експертний висновок результатів перевірки кваліфікаційної роботи на відповідність оформлення вимогам ДСТУ 3008: 2015156 82

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

BSON – binary javascript object notation;

IP – internet protocol;

JSON – javascript object notation;

MFS – minimum feature set

MVP – minimum viable product

TCP – transmission control protocol

UCD –

UI – User Interface

UI/UX

БД – База даних;

ПЗ – Програмне забезпечення;

ПП – Програмний продукт.

ВСТУП

Сучасний світ будується навколо інформації. Інформація повинна оброблятись якомога швидше та якісно, особливо в інформаційно-значимих ресурсах. Саме до таких ресурсів слід віднести процес розробки та реалізації програмного проекту, що є метою професійної діяльності команд розробників програмного забезпечення.

Користувацький інтерфейс – це невід'ємна складова більшості програмних засобів. Як правило, користувачі оцінюють програмне середовище в цілому та його «придатність» до використання за зручністю користувальницького інтерфейсу і в якій мірі він відповідає їх вимогам. За цими вимогами користувачі можуть приховувати різні бажання та визначати як якісні переваги та особисті побажання, так і призначення інтерфейсів, особливості представлення інформації в різних сферах, умови та сервіс використання.

Сподівання та очікування з боку користувачів відносно результатів розробки програмного забезпечення залежать не лише від якісного коду – досить часто цікавий програмний проект може «провалитись» лише за рахунок недостатньої уваги з боку розробників до дизайну користувальницького інтерфейсу. Сучасне програмне забезпечення створюється крос-функціональними командами розробників, до складу яких входять як програмісти і тестувальники, так і дизайнери. За останні роки попит на дизайнерів користувальницьких інтерфейсів став стрімко зростати.

Скорочення UI утворене від англійського «User Interface Design» і дослівно означає – дизайн інтерфейсу користувача. UI-дизайнер відповідає за зовнішнє оформлення продукту та його інтерактивність. Ще декілька років тому від дизайнера UI/UX вимагалось вміння продумати дії користувача і грамотно побудувати юзабіліті сайту або програми. На теперішній час тренд пішов далеко вперед – з'явилося поняття Interaction-дизайнера – тобто людини, яка відповідає за юзабіліті, красу, простоту та читабельність, вміє розуміти і відповідати бізнес-цілям замовника і відстежувати всі останні тенденції. Саме тому ця професія набирає обертів у світі, в тому числі і в Україні.

Активний інтерес в професійному співтоваристві розробників ПЗ викликають кілька тенденцій. По-перше, це алгоритмічний дизайн – як використовувати алгоритми для побудови інтерфейсу, підготовки допоміжного графічного оформлення і контенту, а також персоналізації. По-друге, автомобільні інтерфейси – як зробити їх більш зручними та сучасними. По-третє, дизайн системи – як уніфікувати дизайн лінійки продуктів таким чином, щоб підвищити їх якість і прискорити вихід на ринок. По-четверте, це голосові інтерфейси. І, по-п'яте – змішана, доповнена або віртуальна реальність.

Основною проблемою залишається проектування інтерактивних інтерфейсів складного за своїм функціональним наповненням ПЗ, розробленого для подальшого використання непрофесійними користувачами. Результат проектування, розробки та модифікації користувацького інтерфейсу досить часто виявляється надзвичайно складним та пов'язаним із значними витратами часу. За оцінкою різних спеціалістів на розробку користувацьких інтерфейсів витрачається не менше за половину часу, який необхідно витратити на розробку ПЗ.

Сучасні технології розробки ПЗ залишають значний ступінь свободи при виборі різних підходів до управління колективом розробників та організації їх тісної взаємодії. Разом з тим слід зазначити, що організація співпраці між розробниками ти дизайнерами навіть в межах одного проекту ще далека від досконалості.

У зв'язку з вищенаведеним тема дослідження є актуальною, а результати дослідження та запропоновані програмні рішення будуть спрямовані на зменшення трудовитрат на проектування, прототипування, реалізацію та адаптацію користувацького інтерфейсу ПЗ.

1 АНАЛІЗ ПРОБЛЕМНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Поняття комунікації

Під поняттям комунікації в широкому сенсі цього слова розглядають процеси соціальної взаємодії, що взяті в їх знаковому контексті. Звідси комунікація може бути визначена як передача не просто інформації, а її значення або сенсу за допомогою символів [1].

Взаємодіючи один з одним, включаючись в процес спілкування, люди зазвичай переслідують конкретні цілі. До основних цілей комунікації можна віднести наступні:

– забезпечення ефективного обміну інформацією (прийом і передача) між суб'єктами і об'єктами управління;

– вдосконалення міжособистісних відносин в процесі обміну інформацією;

– створення інформаційних каналів для обміну інформацією між окремими працівниками та групами, координації їх завдань і дій;

– регулювання і раціоналізація інформаційних потоків в рамках організації і за її межами;

– формування умінь і навичок для успішної діяльності;

– формування ставлення до себе, до інших людей і до суспільства в цілому.

В процесі обміну інформацією можна виділити чотири базових елементи [1]:

а) відправник – особа, що генерує ідею або відбирає інформацію для передачі;

б) повідомлення – власне інформація, закодована в символи;

в) канал – засіб передачі інформації;

г) одержувач – особа, якій призначена інформація і яка її інтерпретує.

Ефективна організація діяльності – це така організація роботи, яка дає конкретний позитивний результат – результат як досягнення певної мети. Саме тому на ефективність комунікаційної взаємодії суттєво впливає кількість комунікаційних каналів і «якість» виконання спільних робіт, що вимагають зусиль різних підрозділів компанії одночасно. Разом з тим своєчасність поширення інформації та адекватність застосування інформаційних каналів в залежності від розмірів бізнесу та сфери

діяльності виступає в якості важливих критеріїв оцінки ефективності системи внутрішніх комунікацій.

Виходячи з цього можна сказати, що формування ефективних комунікацій в організації досягається декількома основними способами [2]:

– інформаційними: застосування корпоративних каналів, розсилка повідомлень, дошка оголошень, внутрішній сайт;

– аналітичними: анкетування, фокус-групи, поштові скриньки, опитування;

– комунікативними: корпоративні заходи, змагання за професією, корпоративні тренінги, корпоративне навчання;

– організаційними: розробка і впровадження корпоративних стандартів, виступи керівництва, наради, збори.

Підвищення ефективності комунікацій в середині організації можливо лише на основі детального аналізу використання технологій комунікацій та виявлення існуючих недоліків в комунікаційному процесі в середині організації. Управління та регулювання інформаційними потоками є глобальною проблемою: інформація та повідомлення не завжди потрапляють саме до тієї особи, до якого вони спрямовані, а також – не завжди своєчасно. Тому найбільш часто для вдосконалення комунікаційного процесу та підвищення його ефективності виступає оптимізація системи зворотного зв’язку, усунення комунікаційних бар’єрів, підвищення точності переданої інформації [3]. Саме тому в кваліфікаційній роботі увагу приділено шляхам покращення якості інформаційної взаємодії між учасниками процесу розробки програмного забезпечення – UI дизайнерами та розробниками програмного коду.

1.2 Роль дизайнерів UI в процесі розробки ПЗ

Сучасні користувачі програмних систем звикли до цифрових додатків на мобільних пристроях, які у своїй більшості мають зручний та лаконічний інтерфейс, що підвищило рівень вимог до всіх інших видів користувальницьких інтерфейсів

(UI). Дизайнери стають все більш важливою частиною процесу розробки програмного продукту. Однак співпраця між розробниками і дизайнерами досить часто далека від досконалості.

У своїй більшості розробники ПЗ розглядають функціональні властивості ПЗ відокремлено від користувацького інтерфейсу. Однак користувачі програмного забезпечення не розділяють функціональність та користувацький інтерфейс. Для них саме користувацький інтерфейс і є самою програмою, а наявність зручного інтерфейсу означає позитивне ставлення до усього ПЗ.

Досить часто користувацький інтерфейс розглядає розробниками ПЗ лише як красивий дизайн. Насправді користувач сприймає через інтерфейс всю структуру ПЗ в цілому, користувацький інтерфейс включає в себе всі аспекти дизайну, які впливають на взаємодію користувача та системи. Це не лише той екран, який бачить користувач.

Інтерфейс користувача складається із значної кількості складових, до яких можна віднести:

– набір завдань користувача, який він вирішує за допомогою ПЗ;

– використання системної метаформ (наприклад, робочий стіл в MS Windows);

– елементи управління системою;

– навігація;

– дизайн інтерфейсу тощо.

UI-дизайн (User Interface Design) повинен описувати дизайн графічної структури ПЗ або сайту. Під дизайном інтерфейсу користувача розуміють комплексний процес візуального управління користувачем за допомогою інтерактивного взаємодії між людиною та елементами ПЗ: натискання кнопок, прочитування текстових блоків, перегляд зображень тощо. Саме тому дуже важливо, щоб усі візуальні компоненти знаходились як в естетичному, так і в цільовому сполученні.

Дизайнери інтерфейсу відповідають за те, як у підсумку буде виглядати продукт: вони приймають рішення про макети екранів, кольорові схеми, інфографіку, текстури, віджети тощо.

Крім графіки дизайн інтерфейсу користувача розглядає розробку анімацій, переходів та мікровзаємодій (наприклад, відкликів при натисканні кнопок). Важливо, щоб інтерфейс продукту не містив значної кількості слів, а його використання здійснювалось в основному за допомогою візуальних елементів (наприклад, зображення кнопки, при натисканні на яку з'являється додаткова інформація). Мета розробки сучасного та елегантного UI полягає не просто у створенні естетично приємного, привабливого інтерфейсу, але і у формуванні у користувачів позитивних реакцій після його використання, їх бажання повертатися до цього продукту знову і знову.

До задач, що вирішує дизайнер UI, також відноситься задача забезпечення адаптивність інтерфейсу, що забезпечує максимальну зручність при роботі з ним на будь-яких пристроях.

На рис. 1.1 наведено схему форм, типів та засобів проектування інтерфейсів користувача ПЗ [4].

Рис.1.1 – Класифікація користувацьких інтерфейсів

Під час процесу розробки ПЗ, особливо його видимої частини – інтерфейсу користувача – постає необхідність швидкого, але водночас зваженого вибору способу організації «людино-машинної» та «машино-людинної» взаємодії. Незважаючи на той факт, що більшість розробників ПЗ використовують гнучкі процеси розробки, етап проектування ПЗ досить часто передує етапу розробки інтерфейсу, оскільки саме він служить основою для реалізації користувальницького інтерфейсу. Це призводить до того, що розробники та дизайнери у своїй більшості працюють ізольовано, за виключенням ручного проектування, при якому розробнику надається кінцева версія проекту інтерфейсу у вигляді статичного документа, в якому наведено опис призначеного для користувача інтерфейсу. Деякі фахівці вважають такий підхід помилковим та пропонують розглядати ручне проектування як безперервний спільний процес, а не як разову процедуру, хоча цього може бути важко досягти. Через відсутність постійної співпраці між дизайнером UI та розробником ПЗ реалізація може відрізнятися від дизайну – або через непорозуміння вимог, або через неправильне тлумачення специфікації дизайну. Розробники також можуть бути змушені приймати власні, неузгоджені з дизайнером, проектні рішення в результаті зміни вимог, що не покриваються оновленим дизайном [5].

Запобігти появі помилок в інтерфейсі програмного продукту є нетривіальною задачею. Причиною є той факт, що проектна специфікація не є ідеальним описом програмного забезпечення, яке розробляється, наприклад, через обмеження ресурсів. В роботі [6] наведено у якості прикладу випадок, коли дизайнер розробив лише одну версію для настільного комп'ютера та одну мобільну версію з фіксованою розподільною здатністю, залишаючи всі інші розміри екрану на розсуд розробників. Цей приклад свідчить, що програмістам досить часто доводиться приймати власні дизайнерські рішення або звертатися за допомогою до дизайнера в тих частинах проекту, де дизайн відсутній.

Незважаючи на той факт, що існує значна кількість інструментальних засобів підтримки проектної розробки, використання яких дозволяє допомогти в більш чіткому тлумаченні специфікації, вони не повністю усувають необхідність в прийнятті дизайнерсько-проектних рішень саме розробниками, оскільки дизайн не є

ідеальним уявленням програмного забезпечення. Крім того, такі інструментальні засоби не допомагають проводити постійну перевірку різних версій реалізації проекту в процесі його розробки.

При ідеальному підході до реалізації проекту дизайнери постійно та повністю беруть участь в процесі програмної реалізації, що допомагає розробникам приймати правильні дизайнерські рішення. Але на практиці це виявляється неможливим через ряд факторів: наприклад, ручна перевірка тестової версії програмного забезпечення на предмет наявності змін в дизайні – досить трудомістке завдання. Більш того, дизайнери можуть не володіти необхідними технічними знаннями для регулярної збірки та запуску програмного забезпечення. Хоча розробники обізнані про зміни в програмному забезпеченні за допомогою контролю версій і проводять регулярні перевірки коду, в такій практиці дизайнери зазвичай не приймають участь. Отже, розробники можуть не знати про нові зміни призначеного для користувача інтерфейсу, якщо вони регулярно не оцінюють програмне забезпечення. Ця ситуація є неефективною і призводить до ризику того, що потенційні відхилення від наміченої архітектури ПЗ або інші проблеми залишаться непоміченими.

1.3. Аналіз існуючих рішень

Під інтерфейсом користувача розуміють такий різновид інтерфейсів, у якому одна із сторін інформаційної взаємодії є людиною (користувачем), а друга – технічний засіб. Також він є сукупністю засобів та методів, за допомогою яких користувач взаємодіє зі складною множиною різноманітних елементів. Практично всі види UI під час реалізації проектних рішень з розробки ПЗ зараз ведуться не автоматично і слабо автоматизовані. Процес проектування моделей користувальницьких інтерфейсів є результатом або інтуїтивного підходу програмістів до програмування функціоналу ПЗ, або результатом спільного консультування дизайнерів UI та розробників ПЗ [4].

В результаті проведеного аналізу досліджень, в яких обговорюється співпраця дизайнерів UI та розробників, рішення, запропоновані авторами, різняться. Більшість робіт можна поділити на дві категорії: практичні шляхи, пов'язані з реорганізацією управлінських процесів, і методи, пов'язані з використанням інструментальних засобів, націлених на покращення співпраці дизайнерів і розробників.

Принципи застосування інструментальних засобів у своїй більшості полягають або у наведенні більш детального опису специфікації дизайну користувальницького інтерфейсу, або в наданні розробнику повного доступу до кінцевої реалізації користувальницького інтерфейсу.

Прикладом інструментального засобу, який призначений для вдосконалення специфікацій дизайну, є Enact [6]. Інструмент, зокрема, створений для спрощення співпраці у сфері розробки інтерфейсів сенсорної взаємодії за допомогою жестів, які виявилися досить складними для опису та розуміння розробником. Цей інструментарій дозволяє зменшити кількість помилок в проектах під час їх реалізації за рахунок більш тривалої інформаційної взаємодії, передбаченої проектом – наприклад, за допомогою інтерактивних прикладів. В роботі [7] описаний інструментальний засіб Montage, який допомагає дизайнерам знімати прототипи відео для більш зручного взаємодії. У загальному підсумку можна сказати, що інструменти, які спрямовані на зменшення кількості помилок при розробці користувальницького інтерфейсу, допомагають виявляти помилки якомога раніше, але не намагаються їх повністю уникнути.

Конструювання користувацьких інтерфейсів на основі моделей – це ще один підхід до проектування користувацького інтерфейсу, при якому розробник визначає моделі, які використовуються для безпосередньої генерації реалізованого користувацького інтерфейсу. Цей метод містить кілька переваг, серед яких у першу чергу слід зазначити легку адаптацію до нових вимог. Однак цей підхід також висуває високі вимоги щодо рівня підготовки розробників та суворі обмеження на отримані користувацькі інтерфейси [8]. В роботі підкреслюється, що розробка користувальницьких інтерфейсів на основі моделей має значні переваги в тому, що

воно адаптивне і залежить від контексту у тому змісті, що користувацький інтерфейс може автоматично адаптуватися як до потреб користувачів, так і до платформи.

Деякі науковці та практики [9, 10] відмічають, що дизайнери та розробники традиційно керуються різними процесами. Сучасні дизайнери у своїй більшості приєднуються до підходу, що орієнтований на користувача (user-centered design, UCD), який фокусується на формуванні вимог до ПЗ відповідно до вимог кожного з можливих користувачів, щоб розробити рішення, які інтерактивно задовольняють їх потреби.

Розробка нових функцій ПЗ вимагає тісної співпраці між дизайнером та розробником. Дизайнер повинен спроектувати користувацький інтерфейс заздалегідь для того, щоб розробник міг його програмно реалізувати. Таким чином, встановлюється двоспрямований зв’язок – розробник залежить від дизайнера, щоб повністю реалізувати функції, передбачені в ПЗ, і навпаки. Але оскільки процес розробки ПЗ залежить від термінів календарного планування (особливо в методологіях швидкої розробки ПЗ), може виникати проблема, пов'язана з плануванням часу: дизайнер повинен хоча б на одному спринті випереджати команду розробників [11].

Разом з тим деякі практичні фахівці вважають цей підхід хибним – вони вважають, що краще замінити його більш тісною співпрацею груп розробників та дизайнерів [12]. Вони поясняють, що дизайнер повинен бути постійно доступним для розробників, щоб вирішити всі виникаючі проблеми, пов'язані з дизайном.

1.4 Сучасні концепції побудови UI

За всю історію створення ПЗ розробниками були реалізовані різні способи взаємодії з комп’ютером, втілені у формі призначених для користувача інтерфейсів. Стрімкий технологічний прогрес стимулює появлення нових інтерфейсних рішень.

Серед сучасних підходів до побудови користувацьких інтерфейсів в розробці ПЗ слід виділити наступні їх види:

– невидимий або нульовий інтерфейс (Invisible User Interface, ZeroUI/NoUI);

– матеріальний інтерфейс (Tangible User Interface);

– діалоговий інтерфейс (conversational interface);

– соціальний інтерфейс (Social Interface) тощо.

Поряд з ними зараз активно використовується поняття «природний інтерфейс» (Natural User Interface - NUI), що позиціонується як найбільш оптимальний шлях організації взаємодії з технологією; при цьому були виділені дві основні характеристики, якими, на думку дослідників людино-машинної взаємодії, повинен володіти природний користувацький інтерфейс:

– відсутність додаткових пристроїв при взаємодії;

– інтуїтивність: використання існуючих навичок, мінімізація навчання для використання інтерфейсу

Соціальний інтерфейс (Social Interface) отримав широке поширення в рамках активно розвиваються напрямків соціальних робототехніків, його називають найбільш естетичним способом взаємодії з технологією. Соціалізований користувацький інтерфейс повинен бути орієнтований на поведінку людини. У своїй більшості складовою соціалізованих користувацьких інтерфейсів є програмні інтелектуальні агенти (помічники), головною задачею яких є приховування внутрішньої складності структури ПЗ від кінцевого користувача. Тому сутність проектування такого інтерфейсу полягає у створенні для користувача враження спілкування не з комп’ютером, а із живою людиною (рис. 1.2).

До простих природних інтерфейсів можна також віднести технології взаємодії з одним із комунікативних каналів людини – наприклад, звуковим. Прикладом такого інтерфейсу є голосовий користувацький інтерфейс (голосовий інтерфейс користувача), що дозволяє здійснювати взаємодію з ПЗ за допомогою людської мови.

Рис. 1.2 – Соціальний голосовий інтерфейс

Візуальні та тактильні канали комунікацій становлять основу жестового інтерфейсу користувача (Gesture User Interface) (рис. 1.3), який організує взаємодію за допомогою жестів. Всі жестові інтерфейси можна поділити на три групи:

– інтерфейси, що використовують сенсорні технології;

– безконтактні інтерфейси, основані на рухах людини;

– безконтактні інтерфейси, які використовують додаткові технології.

Рис. 1.3 – Інтерфейс доповненої реальності light-touch

Концепцію матеріального інтерфейсу (Tangible User Interface) було розроблено професором Массачусетського технологічного інституту Хіросі Ісії в 90-х роках минулого сторіччя. На його думку традиційні графічні інтерфейси (GUI) обмежують можливості взаємодії людини з технологіями. Хіросі Ісії вважає, що інформаційні

технології позбавили людини тактильного контакту з оточуючим фізичним світом, тому необхідно знайти можливість природної взаємодії з цифровим мікрофоном, а матеріальне подання цифрової інформації може надавати таку можливість [8]. Такий тип інтерфейсу передбачає використання фізичних предметів та конструкцій; прикладом може служити проект SandScape, в якому користувачі змінюють форму ландшафтної моделі, маніпулюючи піском (рис.1.4) [12].

Рис. 1.4 – Пристрій «SandScape» для навчальної гри у формування ландшафту

Невидимий користувальницький інтерфейс (Invisible User Interface), або, як його ще називають, «нульовий» (ZeroUI/NoUI), в науковому середовищі представляє пік розвитку теорії інтерфейсів. Суть цієї технології полягає у відсутності інтерфейсу – повністю ліквідуються екрани та матеріальні технологічні способи управління (кнопки, пульти тощо). Дизайнери Jakob Nielsen та Rolf Molich у своїй роботі [14] критикують сучасні інтерфейсні рішення, які, за їх думкою, обумовлюють вид взаємодії. Дослідники впроваджують ідею застосування невидимих інтерфейсів в повному обсязі для будь-якого ПЗ, оскільки, на їх думку, саме до такого виду взаємодії повинні наближуватись дизайнери та проектувальники інтерфейсів [14].

Таким чином у підсумку можна зазначити, що складність та багатогранність сучасних методів розробки інтерфейсної взаємодії між користувачем та ПЗ суттєво впливає на увесь процес її розробки та вимагає від колективів розробників або залучати до роботи UI-дизайнерів, які обізнані в побудові програмного коду, або

забезпечити більш тісну взаємодію між ними та іншими учасниками команди розробників.

1.5 Постановка задачі дослідження

Базуючись на проведеному аналізі предметної області, враховуючи стрімкий розвиток технологій розробки інтерфейсів ПЗ, в роботі пропонується проведення досліджень шляхів покращення комунікаційної взаємодії між UI дизайнерами та розробниками ПЗ. З цією метою в роботі було визначено об’єкт дослідження.

Об’єктом дослідження є методи організації комунікації між дизайнерами та розробниками в межах колективів розробників ПЗ.

Метою роботи є проведення досліджень для виявлення шляхів підвищення ефективності комунікаційної взаємодії між учасниками розробки ПЗ за рахунок використання додаткових інструментальних засобів інформаційної взаємодії – чат-ботів.

Для досягнення мети дослідження в роботі необхідно:

– провести дослідження процесів, що супроводжують інформаційну взаємодію між дизайнерами та розробниками;

− проаналізувати використання інструментальних засобів забезпечення інформаційної взаємодії в командах розробників;

– запропонувати власний інструментальний підхід, який забезпечить підвищення ефективності використання засобів інформаційної комунікації між дизайнерами та розробниками;

– розробити модель програмної системи, визначити вимоги до апаратного та програмного забезпечення, виконати проектування та програмну реалізацію системи

– дослідити отримані результати та зробити висновки.

В роботі досліджується методика покращення комунікаційних зв’язків між окремими членами команд розробників ПЗ – дизайнерами UI та розробниками коду –

за рахунок застосування додаткового інструментального ПЗ, оскільки саме їх плідна комунікація дозволить значно покращити якість розробки сучасного ПЗ. В якості такого інструментального рішення в роботі пропонується використання чат-ботів, які на теперішній час мають усі ознаки повноцінного програмного забезпечення.

Для покращення ефективності комунікаційної взаємодії між учасниками процесу розробки ПЗ в роботі планується використання засобів автоматизації аналізу графічних зображень за рахунок впровадження алгоритмів комп’ютерного зору в процес порівняння скріншотів з дизайнерськими рішеннями. Це дозволить зменшити інформаційну відстань між внесенням змін в код ПЗ та узгодженням отриманих результатів в користувальницьких інтерфейсах з боку UI дизайнерів.

2 РОЛЬ ДИЗАЙНЕРА UI В КОМАНДНІЙ РОЗРОБЦІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Підходи до розробки ПЗ

Сучасна індустрія розробки ПЗ розрізняє наступні форми, які визначають підхід до розробки програмного забезпечення:

– принципи;

– моделі;

– методології.

Позначимо основні відмінності цих форм.

Принципи та моделі програмного забезпечення – це набір визначаючих правил верхнього рівня або практик нижнього рівня, положень, на основі яких розроблено велику кількість методологій. До принципів, наприклад, відноситься відомий Agile (гнучка розробка ПЗ) – набір визначених цінностей, принципів та практик, освітніх «гнучких» підходів розробників ПЗ. Це не якась конкретна методологія – це великий набір ефективних практик, які можуть бути використані як у сукупності, так і за окремими ознаками.

Методологія програмного забезпечення – це приватні практики, що базуються на тих чи інших моделях або принципах, що містять конкретний опис дій у процесах розробки ПЗ. Досить часто організація намагається створювати та анонсувати свою власну корпоративну методологію розробки ПЗ, яка буде відрізнятися лише окремим конкретним критерієм, тому на сьогоднішній момент створено величезну кількість різноманітних методологій.

Серед найпоширених методологій найбільшу популярність отримали ті, що засновані на принципах гнучкої розробки.

2.2 Гнучка розробка програмного забезпечення

Agile – це є метод розробки ПЗ, який передбачає швидкий рух та реабілітацію. Багато фахівців у галузі гнучких методологій вважають Agile навіть не фреймворком, а філософією. Однак, якщо говорити про методологію в рамках підходу до управління командою розробників ПЗ, вона спирається на наступні принципи роботи:

а) фокусування на потребах клієнтів;

б) спрощення організаційної структури та процесів;

в) робота короткими циклами;

г) активний зворотній зв’язок;

д) підвищення повноважень співробітників;

е) гуманістичний підхід в управлінні.

В Agile на потребах та інтересах клієнтів фокусується не тільки замовник, а й команда, яка працює над створенням програмного продукту. Такий підхід та фокусування дозволяють створювати більш якісні рішення.

Активний зворотній зв'язок є дуже важливим для будь-якого процесу, адже він дозволяє вчасно скорегувати вектор розвитку проекту, виправити та вилучити з продукту помилки, проводити активне експериментування. Для Agile експериментування – це звичайна практика, що використовується системно і при створенні продуктів, і в побудові процесів, і навіть у взаємозв'язках між членами команд. Застосування Agile слід вважати не метою, а шляхом до мети, обираючи який у компанії з’являється можливість постійно привносити щось нове в процеси розробки ПЗ та вдосконалювати їх. Цей рух є нескінченним, оскільки Agile виступає за зміни – адже найчастіше перший реліз продукту і кінцевий результат не схожі. Тут існують поняття MVP (minimum viable product, мінімально життєздатний продукт)), і MFS (minimum feature set, мінімальний набір функціональності). MVP і MFS – це не тільки «каркаси» або основа для більш складних і функціональних рішень, але і «стратегія для отримання конкурентоспроможного продукту і тестування його можливостей» [15].

Найчастіше впровадження нової методології обумовлено так званою SMART-метою. SMART-мета – це [16]:

S – specific: конкретна; significant: значна; stretching: розтягувана.

M – measurable: вимірна; meaningful: повна певного сенсу; motivational: мотивуюча.

A – agreed upon: узгоджена; attainable, achievable: досяжна; action-oriented: орієнтована на дії.

R – realistic: реалістична; relevant: релевантна; reasonable: прийнятна; rewarding: корисна; results-oriented: орієнтована на результати.

T – time-based, time-bound, timely: обмежена в часі; tangible: чутлива; trackable: відстежувана.

В Agile існують свої SMART-цілі. До них відносяться:

– продукти, які відповідають вимогам;

– скорочення часу випуску продукту;

– рання окупність інвестицій;

– гнучкість;

– зменшення ризиків;

– висока прозорість процесу;

– ефективність;

– передбачуваність;

– задовільнення замовників;

– покращення настрою.

Agile за своєю суттю – це «парасолька» (див. рис. 2.1), під якою знаходяться інші методології. Найбільш відомими з них є Lean, Kanban та Scrum. Розглянемо їх більш детально.

Lean вважається попередником Agile і побудований на ощадливому управлінні. Робота такого підходу, з одного боку, вкрай проста. Спочатку знаходять бар'єри, які заважають зростанню і конкурентоспроможності компанії, після чого їх оцінюють: як подолати, обійти, змінити. При цьому планують не одноразові дії, а тривалий процес, до якого залучені як власники бізнесу, так і рядові працівники. Lean будується на

створенні цінностей в компанії і досягненні їх з одночасним зменшенням виробничих витрат. При використанні Lean-менеджменту робота компанії ділиться на операції і процеси, головним завданням яких є або створення цінності, або її виключення. Завданням всього управління стає планомірне виключення тих процесів, які не додають цінності для споживача.

Рис. 2.1 – «Парасолька» Agile

Kanban, як і Lean, відноситься до «бережливого управління». Вперше метод був застосований в автомобільній компанії Toyota в 60-х роках XX сторіччя. У перекладі з японської, кан-бан – це «сигнальна дошка» (рис. 2.2): така дошка використовується для візуалізації наростаючого темпу, що дозволяє давати більше продукції. Співробітники на кожному етапі процесу не можуть перейти до наступної фази роботи, поки за допомогою Kanban-дошки не буде дано відповідний сигнал.

Систему Kanban зручно використовувати, щоб досягти балансу між навантаженням на команду та її здатністю виконувати завдання. До властивостей цього методу входять: візуалізація робочого потоку; визначення робочого навантаження; контроль робочого процесу; конкретизація робочого процесу; сумісна праця тощо.

Kanban-дошка також може містити область зі значенням «в процесі». Така методика дозволяє не тільки контролювати всі етапи, а й поділити велику задачу на підзадачі, виконувати роботу за мірою надходження, а не робити все і відразу, що з

великою ймовірністю закінчується невдачею. Крім того, практикують додавання колонки з назвою «Ідеї» для запису можливих майбутніх завдань.

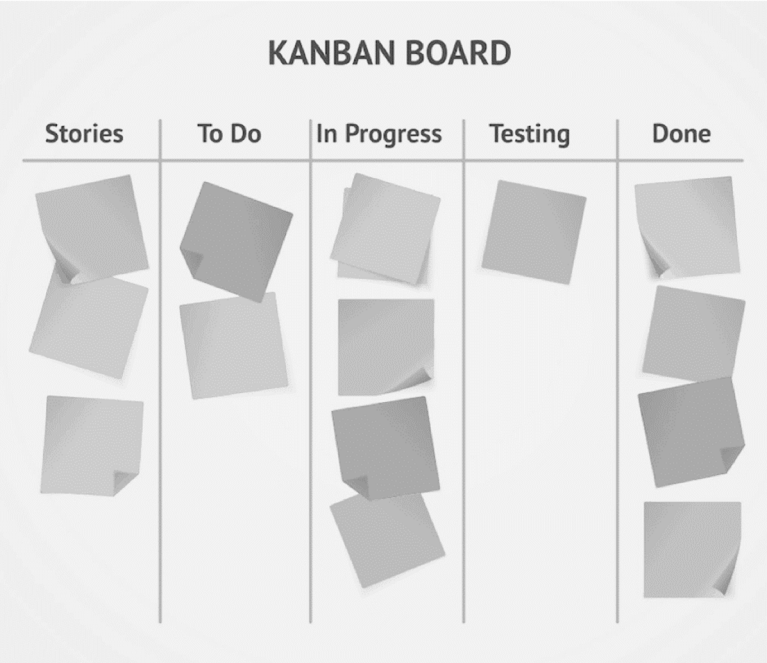

Рис. 2.2 – Приклад Kanban-дошки

Scrum – це термін з гри в регбі, перекладається як «сутичка». Подібний метод гри вимагає зібраності команди, повної взаємодії її членів. Основа методології полягає в об'єднанні принципів «бережливого управління»: «…Коли б не був запущений проект – вам ніщо не заважає регулярно перевіряти хід робіт і послідовно з'ясовувати: справляєтеся ви із завданням; в потрібному напрямку рухаєтеся; створюєте саме те, що насправді хоче отримати замовник; чи є способи вдосконалити методи розробки і виконувати роботу найбільш якісно і швидко»[17]. При цьому якщо процес роботи або продукт не відповідають очікуванням, слід проявити гнучкість і змінити хід робіт. «Кінцевим результатом застосування методології <...> є команди, які наочно збільшують свою продуктивність» [17]. Можна зробити висновок, що Scrum виступає скоріше не як методологія управління проектами, а у якості зібрання принципів організації створення і випуску продукту. Важливо розуміти, що, на відміну від описаних раніш методологій, в Scrum збирається нова команда, головні ролі в якій відводяться Власнику продукту та Скрам-майстру.

Власник продукту, або Product Owner, самостійно вирішує, як буде виглядати кінцевий продукт і в чому полягає суть проекту. Він єдиний в команді, хто чітко розуміє правила бізнесу, в якому працює.

Скрам-майстер відповідальний за всю роботу колективу. Він є сполучною ланкою між командою та Власником Продукту.

Команда розробки - основний двигун в роботі. Учасники мають суміжні навички, що дозволяє підхоплювати роботу один одного в разі стагнації проекту. При цьому кількість людей в Scrum-команді – від 6 до 8, оскільки вважається, що чим більше людей – тим складніше ними управляти, і це є поганим для проекту. Спільне проектування створює єдиний інформаційний простір між розробниками і дизайнерами, що дозволяє поліпшити розуміння нового продукту всіма учасниками.

Таким чином можна зробити висновки, що всі сучасні технології швидкої розробки ПЗ передбачають тісну інформаційну взаємодію між усіма учасниками команди.

2.3 Роль та значення UI дизайнера в команді

Враховуючі вище наведені принципи швидкої колективної розробки слід зазначити, що в ідеальному випадку UI дизайнер повинен супроводжувати увесь процес розробки ПЗ – від етапу постановки задачі до передачі його замовнику, а також мати суміжні навички щоб допомагати учасникам проекту у разі необхідності. Однак в реальному житті все змінюється: дизайнер UI у своїй більшості – творча людина, яка може не володіти навичками програміста або тестувальника, та не мати необхідних технічних знань для регулярної збірки і запуску нових версій програмного продукту.

Проекти, якими займаються розробники, досить часто мають вже чітко визначені вимоги, такі як дизайн-прототип або функціональні вимоги, що описані в технічному завданні. Фірма зазвичай дотримується методології гнучкої розробки

програмного забезпечення, але оскільки проекти розрізняються за масштабом і потребами, вони відповідним чином адаптують процес. Деякі невеликі проекти можуть складатися тільки з одного дизайнера і одного розробника та виконуватись в рамках одного спринта. В таких проектах зазвичай використовується менш формальний процес, оскільки дизайнер і розробник можуть тісно співпрацювати. Більш великі проекти можуть складатися з невеликої групи людей і можуть тривати кілька місяців з декількома спинтами протягом більш тривалого періоду. У таких проектах часто використовується більш строгий процес, який включає стандартні гнучкі методи, такі як щоденні наради-звіти і демонстрації спринтів.

Хоча дизайнери UI безумовно є важливою частиною процесу розробки ПЗ, , але дизайнери та розробники традиційно дотримуються різних процесів. Особливості спільного проектування можна простежити в різних підходах до проектування, орієнтованого на людину.

Найчастіше сучасні дизайнери дотримуються підходу, орієнтованого на людину-користувача (human-centred design, HCD), який фокусується на формуванні співчуття до кінцевого користувача та дозволяють розробляти рішення, які ітеративно задовольняють їх потреби. Багато видів діяльності в HCD включають контакт з кінцевим користувачем, наприклад, щоб зрозуміти проблему або перевірити, чи задовольняє їх вимогам запропонований дизайн.

Залучення клієнтів, споживачів і користувачів в розробку програмного продукту обумовлюють основні дії в процесі реалізації підходу, орієнтованого на людину-користувача, і організовані за допомогою ітераційних циклів (рис. 2.3).

В роботі [10] наведено результати дослідження методів співпраці дизайнерів і розробників. Автори виявили, що члени команди фізично розташовувались у приміщенні або по професії, або по команді. Командна розсадка мала таку перевагу, як поліпшення комунікації між дизайнерами і розробниками. Деякі вважали за краще сидіти за професією, щоб легше було обговорювати технічні проблеми. Дослідники також виявили, що розробники фізично переміщалися по приміщенню щоб поговорити з дизайнерами тільки тоді, коли їм було потрібно обговорити проблеми, пов'язані з проектуванням та реалізацією UI. Дизайнери UI так і не змогли встановити

контакт з розробниками, крім доставки артефактів дизайну. Саме тому найбільш поширеною практикою їх співпраці на теперішній час стає обмін артефактами: дизайнер створює артефакт дизайну до початку програмної реалізації. Готовий артефакт дизайну передається розробнику, який приступає до реалізації призначеного для користувача інтерфейсу. В дослідженні [10] підкреслюється, що на цьому етапі розробки ПЗ співпраця між розробниками та дизайнерами майже наближена до нуля.

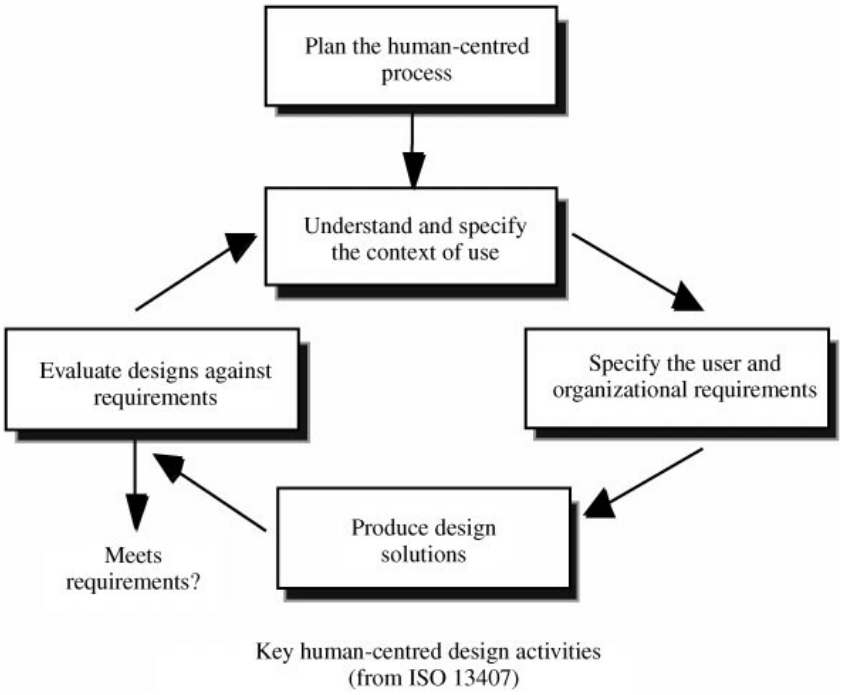

Рис. 2.3 – Проектування UI дизайну, орієнтованого на людину.

Артефакти дизайну можна розглядати як схему, яка описує різні аспекти кінцевої версії користувальницького інтерфейсу. Існують різні типи артефактів дизайну: від статичних макетів до більш складних інтерактивних прототипів, які також можуть демонструвати інтерактивну поведінку – таку як навігація між сторінками або анімація. Статичні макети виглядають як знімок екрану з бажаними елементами користувальницького інтерфейсу (див. рис. 2.4), часто їх створюють у векторному форматі з використанням інструментальних засобів дизайну інтерфейсу (наприклад, таких як Sketch та Figma).

За рахунок застосування в процесі розробки величезної кількості інструментів для створення прототипів керівники груп розробників намагалися подолати розрив

між артефактом дизайну та його реалізацією, однак ці інструменти ще не дозволяють створювати програмне забезпечення промислового рівня.

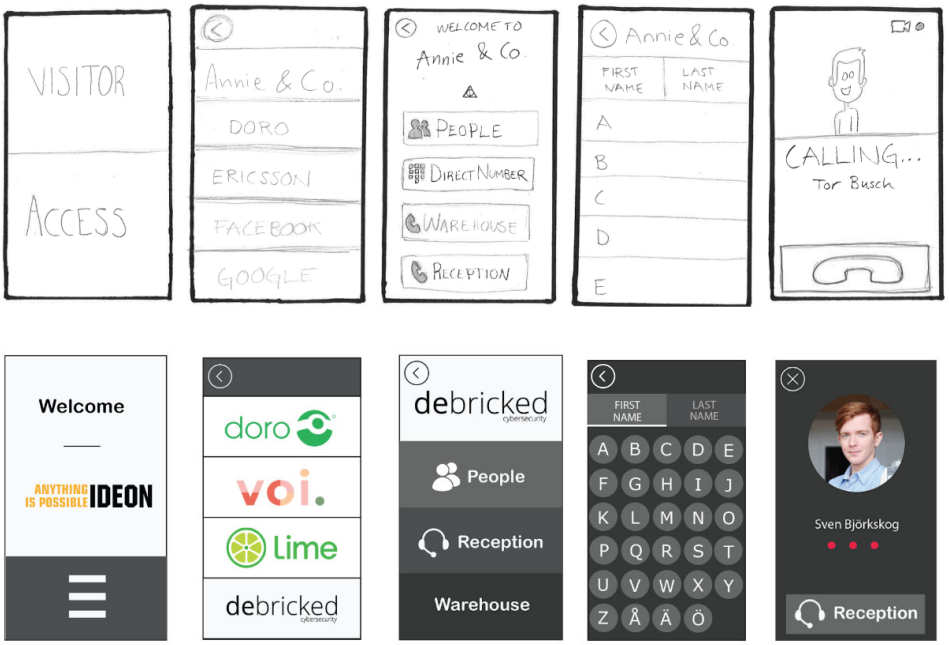

Рис. 2.4 – Приклади артефактів UI дизайну

В результаті опитування будо виявлено, що дизайнери використовують найрізноманітніші інструменти і методи, щоб повідомити розробнику про різні аспекти дизайну – від індивідуального спілкування до візуалізації дій в режимі онлайн. За даними цього ж опитування візуальний дизайн досить часто є пріоритетним [14].

Проаналізувавши вимоги щодо вмінь та навичок дизайнерів UI, які висуваються з боку роботодавців, в роботі [11] було визначено специфічні та загальні вимоги (див. рис.2.5), які впливають на кінцевий результат розробки.

Разом з тим усі дослідники відмічають низький рівень комунікації між дизайнерами та розробниками, які у своїй більшості обмежуються стандартними каналами комунікації, які визначено середовищем підтримки розробки програмного проекту: електронна пошта, заплановані зустрічі та у меншому ступені – чат.

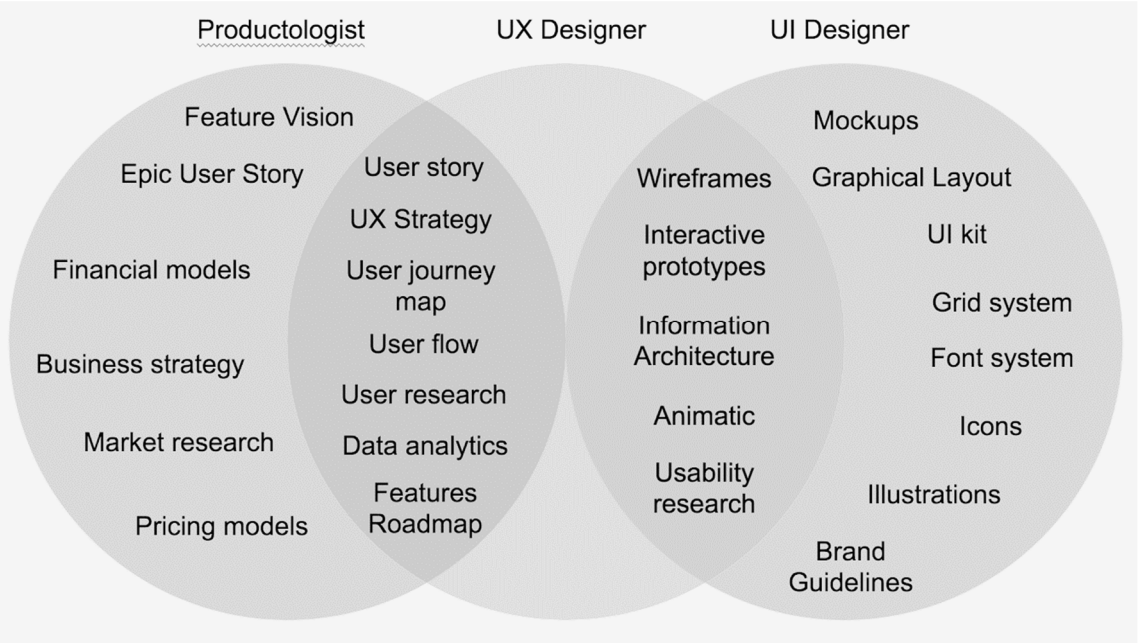

Рис. 2.5 – Поєднання компетенцій UX/UI-дизайнера

Лише окремі команди застосовують спеціалізовані інструментальні засоби розробки: наприклад, Zeplin (https://zeplin.io/), який дозволяє перетворити статичний макет в детальну специфікацію, включаючи розміри і коди кольорів, а іноді – навіть фрагменти коду, що допомагає розробнику більш точно та ефективно виконати кодування інтерфейсу.

3 ТЕХНОЛОГІЇ ЗАБЕЗПЕЧЕННЯ КОМУНІКАЦІЇ

3.1 Необхідність постійної комунікації

Маніфест Agile (2001 р.) містить головні тези цього підходу до розробки програмного забезпечення [16]:

– люди та взаємодія більш важлива ніж процеси та інструменти;

– працездатне програмне забезпечення більш важливе ніж вичерпне документування;

– співпраця із замовником більш важлива, ніж узгодження умов контракту;

– готовність до змін є найважливішим у порівнянні з першопочатковим планом.

Звертаючи увагу на перший постулат Маніфесту ми приходимо до висновку, що засоби забезпечення комунікаційної взаємодії між учасниками проекту виходять на перший план при реалізації проектних рішень з розробки ПЗ.

Комунікації – невід'ємний елемент, що діє в будь-яких командах. Особливо вони важливі при розробці ПЗ. У процесі спілкування формуються загальні соціально-психологічні відносини [1].

Спільна робота в команді вимагає від її членів повного взаєморозуміння, і без спілкування цього отримати неможливо.

Спілкування в команді може бути різним: загальним, пов'язаним з проектом та таким, що дозволяє зняти психологічне напруження. Всі види спілкування необхідно планувати, щоб керувати взаємодією членів команд і формувати внутрішньо-командні відносини співробітництва та взаємодопомоги.

В роботах [5-10] наведено результати аналізу проблем, що виникають під час програмування розробниками інтерфейсів, які вимагають комунікації з дизайнером. Ці проблеми, як свідчать дослідження, стають блокуючими, та можуть бути характеризовані як збій в дизайні UI. Було виявлено три основних категорії причин, які викликають такі недоліки розробки:

а) відсутня необхідна інформація: дизайнер не повністю визначив зовнішній вигляд або порядок взаємодії певних елементів (наприклад, як повинна виглядати кнопка при наведенні на неї курсору);

б) граничні випадки: дизайнер не врахував екстремальних або проблемних ситуацій (наприклад, порожній рядок або дуже довгі імена користувачів);

в) технічні обмеження: розробник не врахував технічні обмеження, що призводять до додаткового часу розробки або складності (наприклад, необхідність реалізації повністю налаштованого компонента користувальницького інтерфейсу, незважаючи на той факт, що в системі присутні кілька схожих та вже готових до використання компонентів користувальницького інтерфейсу).

Деякі дизайнери додатково створюють коригувальні документи, в яких містяться анотовані знімки екрану, щоб повідомити розробнику про особливості програмної реалізації інтерфейсу електронною поштою.

В роботі [18] вказано, що практика організації зворотного зв'язку різниться між різними командами та окремими особами. Наприклад, один з розробників завжди просив висловити свою думку про програмну реалізацію дизайну та інтерфейсів, перш ніж вважати його завершеним. Інший розробник зауважив, що зворотній зв’язок надавався тільки під час ретроспективи спринту.

Для організації інформаційної підтримки проекту в команді зазвичай використовуються наступні методи:

– дошки оголошень для повідомлень та звітів за проектом;

– поштові розсилки за проектом;

– веб-сайт проекту для публікації документів, сторінок членів команд, технічних завдань, змін у проекті, навчальних матеріалів тощо;

– консультування за темою проекту для постановки завдань та обговорення ключових моментів;

– засідання, присвячені обговоренню найкращого досвіду та презентація членами командних рішень складних завдань.

Спілкування з використанням електронної пошти зазвичай використовується у тому випадку, коли спілкування не залежало від термінів часу. У своїй більшості

розробники використовують чат для швидкого розв’язання питань, формування зворотного зв'язку і обміну файлами. Разом з тим до недоліків використання чатів слід віднести той факт, що повідомлення досить часто губляться в історії, а саме спілкування в чаті іноді стає некерованим і за ним важко стежити. Поширеними рішенням для організації корпоративного чату є Slack [19], Microsoft Teams, Skype, Telegram, Rocket Chat, Sygnal та інші.

Slack (рис.3.1) – це онлайн-сервіс для ведення переписки всередині команди. Цей сервіс відрізняється можливістю тісної інтеграції активних діалогів з іншими додатками. Такий принцип дозволяє вести моніторинг прогресу роботи над різними проектами за допомогою єдиного інтерфейсу і прибрати зайву інформацію.

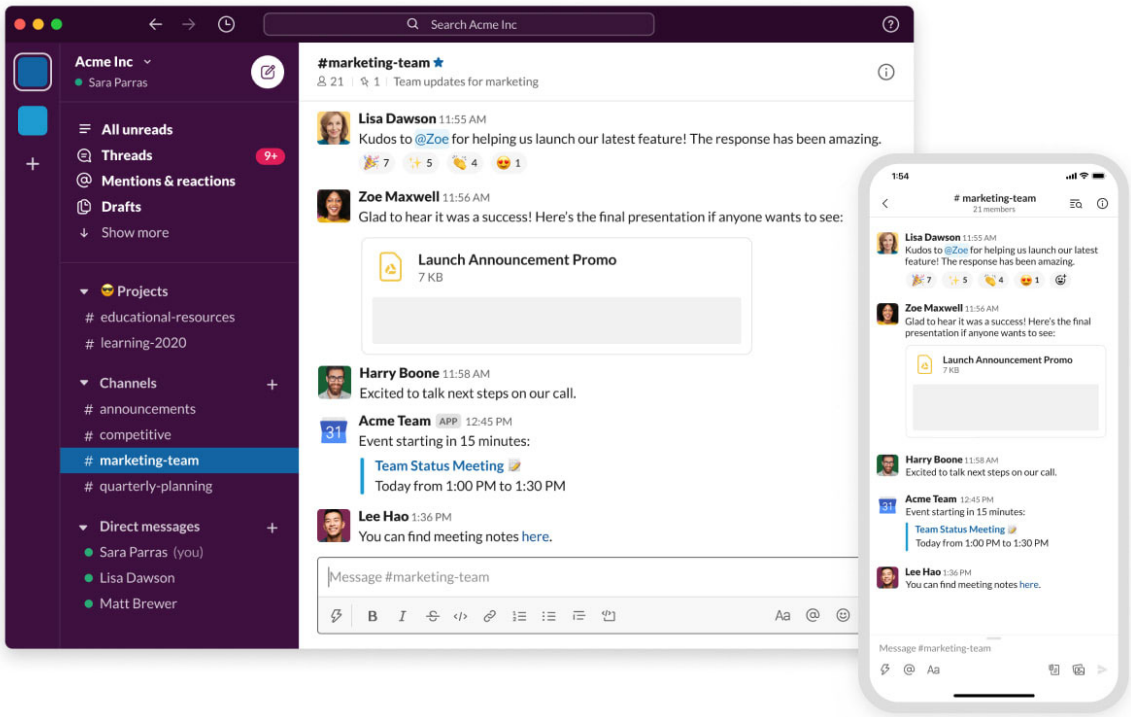

Рис. 3.1 – Чат Slack на різних екранах

До основних переваг використання цієї програми слід віднести [19]:

– обговорення та ухвалення рішень у невеликих (3-10) групах людей;

– вирішення ситуації, коли невідомо, хто саме може відповісти на питання;

– поширення оголошень та апдейтів;

– direct messages співробітникам і відділам;

– автоматичні повідомлення й алерти (значна кількість інтеграцій зі сторонніми сервісами та ботами);

– проведення коротких опитувань (наприклад, обрати час зустрічі);

– можливість організації single channel guest – можливість створення окремих каналів, у які запрошують людину ззовні, наприклад, кандидата, фрілансера або консультанта.

Slack дуже функціональний, цей додаток має інтеграцію з безліччю сервісів: від пошти, хмарних сховищ, органайзерів і сервісів заміток до додатків для медитації та розваг. Серед них також є і спеціалізовані сервіси для підтримки роботи дизайнерів.

Headless-браузер – це браузер без графічного інтерфейсу користувача, який можна запрограмувати для виконання, наприклад, переходу за URL-адресою, натискання кнопки або створення знімка екрану. Прикладами таких систем можуть бути Selenium, який досить часто використовується для автоматичного тестування користувальницьких інтерфейсів, або Puppeteer від Google, який надає можливість дослідження виконання функції в середині Google Chrome. Інтеграція такого сервісу з чатом Slack дозволить покращити результати співпраці з дизайнерами та підняти міжособистісні комунікації на новий рівень.

3.2 Програмний засіб для покращення якості комунікації

Одним з елементів організації інформаційного обміну є бот. Бот – це програмне забезпечення, яке дозволяє виконувати чітко визначені завдання за запитом. В дослідженні було зосереджено увагу на класі ботів, які взаємодіють з користувачами в межах чату – чат-ботах.

Визначення поняття чат-бот наведено в декількох джерелах, але найбільш природним вважається наступне визначення: «Чат-бот – це штучно створена програма, яка може підтримувати дискусію чи розмову з людиною». Таким чином,

чат-бот є елементом інформаційно-комунікаційної системи, яка може відповісти на запит або твердження у зручній для користувача формі.

У своїй більшості чат-боти взаємодіють з користувачами у формі обміну миттєвими повідомленнями, які сформовані та адаптовані за рахунок використання методів штучного інтелекту та машинного навчання. Чат-бот запрограмований на роботу незалежно від наявності людини-оператора. Він може відповідати на запитання, які подані природною мовою, і відповідати як реальна людина, а також може надавати відповіді на основі комбінацій попередньо визначених сценаріїв.

Реакція чат-ботів обумовлена завантаженою в нього базою знань. У разі відсутності в базі знань поняття, яке визначається запитом, він або ігнорує такий запит, або може передати комунікацію людині.

Метою дослідження стала розробка чат-боту для покращення якості комунікації між дизайнерами та розробниками. Чат-бот дозволить дизайнерам відстежувати прогрес в програмній реалізації призначеного для користувача інтерфейсу. Під час спільного використання чат-боту та системи підтримки розробки програмного проекту скріншоти екрану користувальницького інтерфейсу будуть автоматично завантажуватись в чат-бот, коли розробник зафіксує будь-які зміни в репозиторії проекту. Чат-бот орієнтований на визначення розбіжностей між поточним зображенням інтерфейсу у порівнянні з передостанньою версією. У разі, якщо чат-бот фіксує відмінності, то результат буде направлено у відповідний канал чату Slack, де присутні і дизайнери, і розробники.

3.3 Алгоритми порівняння зображень

Зіставлення зображень є одним з фундаментальних аспектів багатьох завдань комп'ютерного зору, таких як розпізнавання об'єктів, побудова тривимірної сцени на основі декількох зображень, створення стереопари або створення панорамних

зображень. На теперішній час не існує певного універсального методу, що ідеально підходить для всіх завдань такого роду.

Людині нескладно порівняти два зображення і виділити на них об'єкти – все це відбувається на інтуїтивному рівні і не викликає ускладнень. Але для комп'ютера зображення – лише набір даних, що не несе інформації про об'єкти, що зображені на ньому. Існують два основні підходи щодо наділення ЕОМ таким умінням.

Перший підхід полягає в порівнянні зображень на основі інформації, що отримана від аналізу всіх пікселів зображення [20]. У загальному випадку алгоритм такого типу працює наступним чином: для кожного пікселя обчислюється значення деякої функції і на основі цих значень формується певна характеристика всього зображення в цілому. Тоді задача порівняння двох чи більше зображень зводиться до порівняння цих характеристик. Хоча такі алгоритми досить прості і не вимагають великих обчислювальних витрат, результат їх роботи залишає бажати кращого. Причина полягає в їх основній ідеї – в характеристику зображення включається інформація щодо кожного пікселю зображення, незважаючи на його цінність. Метод формує неприйнятний результат в разі появи нових об'єктів, перекриття одних об'єктів іншими, появи шуму тощо.

Другий підхід, який було обрано для використання в даній роботі, заснований на ідеї використання лише тих пікселів, внесок яких в загальну характеристику буде значним – зображення замінюється на набір так званих особливих точок (рис.3.2).

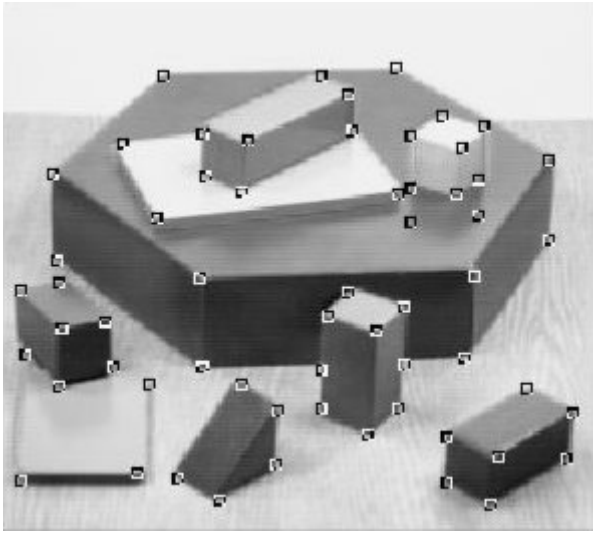

Рис. 3.2 – Особливі точки на зображенні

Особливими називаються такі точки зображення, локальні околиці яких володіють деякими особливостями в порівнянні з околицями інших точок зображення.

Особливі точки мають наступні властивості [21]:

Повторюваність (repeatability) – особлива точка знаходиться в одному і тому ж місці сцени або об'єкта зображення, незважаючи на зміну кута огляду або освітлення спостерігається сцени.

Інформативність (informativeness) – околиці особливих точок значимо відрізняються одна від одної.

Локальність (locality) – особлива точка займає невелику область зображення.

Кількість (quantity) – кількість особливих точок достатньо велика для того, щоб їх вистачило для виявлення навіть невеличких об'єктів.

Точність (accuracy) – виявлені особливі точки точно локалізуються.

Ефективність (efficiency) – особливі точки зображення виявляються за прийнятний час в критичних до часу додатках.

Процес порівняння зображень можна поділити на три етапи.

Перший етап – знаходження множини особливих точок за допомогою методів, які називаються детекторами. Ці методи забезпечують інваріантність знаходження одних і тих же точок щодо перетворень зображення. Разом з тим слід зазначити, що недостатньо використання лише детектора, оскільки результатом його роботи є множина координат особливих точок. Для цього на другому етапі відбувається побудова дескрипторів. Дескриптор – це опис точки, яка унікально ідентифікує її серед множини всіх точок. Дескриптор повинен забезпечувати інваріантність знаходження відповідностей між точками щодо перетворень зображення. Деякі методи виконують відразу обидва завдання – детектування особливих точок і побудова дескрипторів. Третій етап полягає в порівнянні дескрипторів і пошуку точок, що співпадають на обох зображеннях [22].

Одним з найбільш поширених детекторів особливих точок є детектори кутів на зображенні. Зручність їх використання полягає в тому, що кути на зображенні можна однозначно зіставити, на відміну від ребер.

Метод Моравеця [22] вимірює зміну яскравості пікселя *(x, y)* за допомогою зміщення квадратного вікна з центром в *(x, y)* на один піксель в кожному з восьми напрямків (вгору, вниз, вправо, вліво, і в чотирьох напрямках по діагоналі). Розмір вікна найчастіше обирається в 3x3, 5x5 або 9x9 пікселів.

Метод Харріса [23] базується на детекторі Моравеца та є його покращенням. Метод використовує для визначення особливих точок похідну першого порядку від інтенсивності пікселя, що дозволяє визначити зміну яскравості в локальній околиці точки. Результатом роботи методу Харріса є сформована М-матриця, до якої заносяться зміни інтенсивності, що вираховуються окремо для кожного з пікселів зображення (рис. 3.3).

Рис. 3.3 – Детектор кутів Харріса

Досить успішним є алгоритм FAST (Features from Accelerated Segment Test) [23]. В алгоритмі розглядається коло з 16 пікселів навколо точки-кандидата *P*. Точка є кутовою, якщо для поточної розглянутої точки *P* існують *N* суміжних пікселів на окружності, інтенсивності яких більші за $I_P + t$ або інтенсивності всіх менші за $I_P - t$, де I_P – інтенсивність точки *P*, *t* – гранична величина. Далі необхідно порівняти інтенсивність у вертикальних та горизонтальних точках на окружності під номерами 1, 5, 9 і 13 з інтенсивністю в точці *P* (див. рис. 3.4).

Даний алгоритм має ряд недоліків: наприклад, його ефективність залежить від порядку обробки зображення і розподілу пікселів, тому поблизу деякої околиці може виявитися декілька особливих точок. Разом з тим цей метод досить широко використовується для машинного навчання.

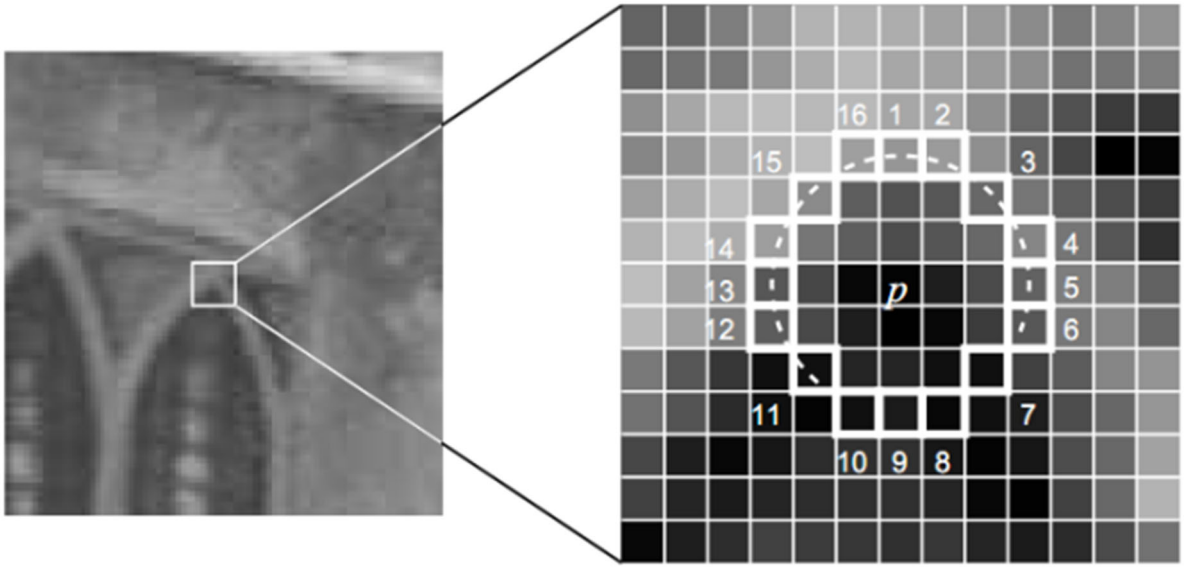

Рис. 3.4 – Робота алгоритму FAST

Ще одним методом для визначення об’єктів можна вважати метод побудови подання сцени (background model). Він базується на пошуку відхилень від базової моделі для кожного нового зображення. Зміни в зоні зображення можуть приймати значні відхилення у порівнянні з базовим зображенням. В цьому методі для моделювання кольору пікселів використовується змішування по Гаусу, а піксель в кожному кадрі порівнюється з моделлю фону [23].

Для розв’язання задачі дослідження щодо реєстрації розбіжностей між скріншотами, які потребують контролю з боку UI дизайнерів, будемо використовувати програмну реалізацію алгоритму Харріса, яка знайшла своє поширення для аналізу статичних зображень. Цей алгоритм достатньо швидкий, а також дозволяє налаштовувати відокремлені варіанти чутливості для різних ділянок зображення, що дозволяє запобігти спрацюванню детектора у разі незначних змін в дизайні інтерфейсу ПЗ. Програмна реалізація алгоритму доступна у вигляді opensource-бібліотеки OpenCV.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ

4.1 Архітектура чат-бота

Архітектура чат-ботів залежить від мети їх розробки. Існують різні причини, з яких людина використовує чат-боти, тому чат-боти можуть бути класифіковані як особисті боти, бот служб обслуговування або функціональні боти. Відповідно до цих категорій архітектура ботів поділяється на два типи моделей: генеративна модель та модель, що заснована на пошуку. Зазвичай генеративна модель використовується для побудови діалогових розумних ботів, які призначені для реалізації складних інтелектуальних алгоритмів (рис.4.1). Ці чат-боти дозволяють організувати відкриту інформаційну взаємодію із середовищем уподібнюючись людині.

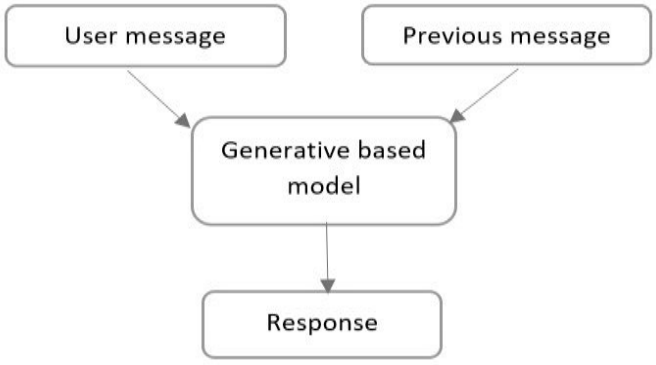

Рис. 4.1 – Генеративна модель чат-боту

Інший підхід використовують при програмуванні чат-ботів, що базуються на моделі, заснованій на пошуку. Ці чат-боти досить легко програмно реалізувати з урахуванням заздалегідь відомого контексту користувача (див. рис.4.2). Головне обмеження полягає в тому, що іноді запитання, які задаються користувачем, не мають свого відображення в існуючій БД питань та відповідей чат-боту.

Розробка архітектури чат-боту – це отримання знань про потреби користувачів. Для досягнення мети дослідження в рамках написання кваліфікаційної роботи щодо покращення якості комунікації між розробниками та UI дизайнерами чат-бот повинен виконувати інформування останніх якомога швидше у тому разі, якщо зміни у

програмному коді одночасно викликали значні зміни в інтерфейсі ПЗ. Інформування може виконуватись декількома способами:

– чат-бот інформує дизайнера про факт внесення змін в дизайн інтерфейсу;

– чат-бот демонструє дизайнеру результат внесення змін в дизайн інтерфейсу за рахунок зміни коду ПЗ;

– чат-бот оцінює ступінь внесених змін та у разі перевищення визначеного критерію інформує дизайнера та розробника про відхилення від затвердженого макету.

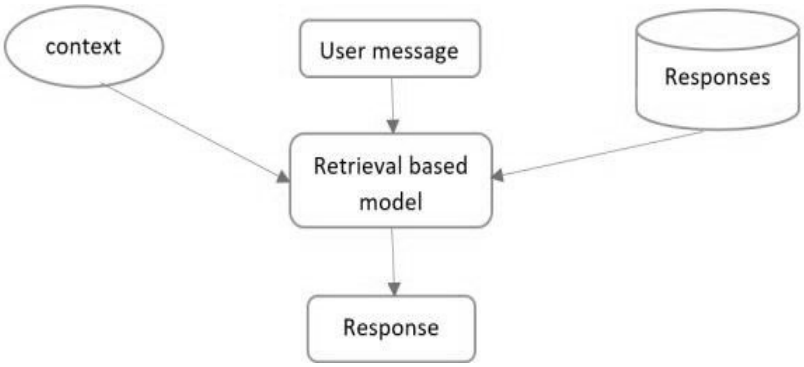

Рис. 4.2 – Модель чат-боту, заснована на пошуку

В якості середовища використання боту було обрано корпоративний месенджер Slack, який має значне поширення серед розробників ПЗ. Для реалізації взаємодії будемо використовувати API Slack для відправки повідомлень в DialogFlow (DF). Інтерфейси Slack і DialogFlow разом формують чат-бота (рис.4.3).

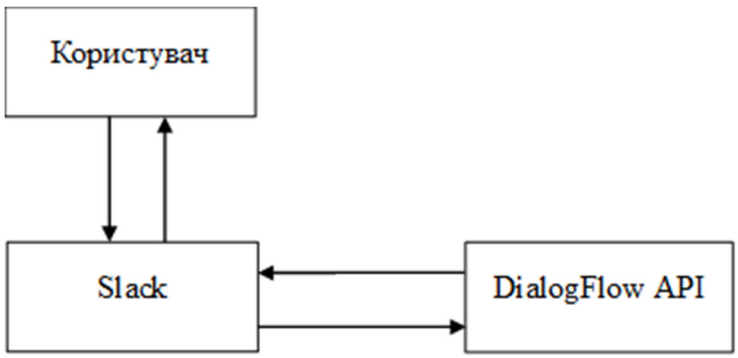

Рис.4.3 – Узагальнена архітектура чат-бота

Чат-бот реалізований за принципом безперервного конвеєру. Безперервний конвеєр постійно взаємодіє з DockerHub для завантаження скриптів, що здійснюють порівняння зображень. Чат-бот підтримується базою даних, яка використовується для зберігання запиту структурованих даних та службою файлового сховища – для зберігання знімків екранів. Сценарій безперервного конвеєра завантажує файли безпосередньо в служби файлового сховища. Доступ до файлів також можна отримати через будь-який зовнішній інтерфейс.

4.2 Вибір засобів програмної реалізації

Чат-боти – це повноцінні програмні системи, орієнтовані на користувача. Саме тому в якості інструментальних засобів для програмної реалізації чат-бота було обрано мову програмування JavaScript та фреймворк React.js. Така інтеграція дозволяє з легкістю будувати чат-боти будь-якої складності.

Мова програмування JavaScript поєднує декілька стилів програмування: об'єктно-орієнтований, імперативний та функціональний. Найбільше поширення знаходить в браузерах як мова сценаріїв для додання інтерактивності веб-сторінок. До основних архітектурних рис JavaScript слід віднести динамічну типізацію, слабку типізацію, автоматичне керування пам'яттю, прототипне програмування тощо [24].

React.js – це бібліотека JavaScript з відкритим кодом, яка досить широко використовується для побудови інтерфейсів, для реалізації обробки запитів програми до будь-якого іншого зовнішнього інтерфейсу – апаратного чи програмного, також вона забезпечує використання компонентів інтерфейсу. React знайшов широке використання при розробці веб-орієнтованого програмного забезпечення, оскільки забезпечує зміну дані на екрані користувача без перезавантаження сторінки веб-додатку. До головних переваг використання React.js слід віднести легке масштабування бібліотеки, простота використання та швидкодія [24].

Бібліотека Slack Bolt [25] адаптує React для написання чат-ботів. Slack Bolt забезпечує розробку чат-ботів через декларативне програмування і компонентний підхід. Чат-бот може надсилати повідомлення у вигляді тексту, зображень, кнопок і багатьох інших UI елементів та реагувати на повідомлення від користувачів. Наприклад, для відправлення текстового повідомлення достатньо викликати функцію message, яка стежить за змінами повідомлень, і у разі отримання відповіді здійснюється виклик функції postMessage, через яку передається id каналу та сам текст, який ми бажаємо відправити (рис.4.4).

```
app.message(answer, listeners: async ({ message : AppRequestedEvent | ... , client : WebClient }) => {

  if (answer) {
    await client.chat.postMessage( options: {
      channel: 'C020B1C5SSH',
      text: `${user} ${message.text}`,
      username: 'Designer',
    })

    answer = '';
  }
});
```

Рис.4.4 – Функція формування текстового повідомлення

Особливістю застосування бібліотеки Slack Bolt є той факт, що аналогічно існує можливість відправити й графічне зображення (рис.4.5), що доцільно використовувати при розробці чат-боту, який є метою кваліфікаційної роботи.

```
app.message(message, listeners: async ({ message : AppRequestedEvent | ... , body : EnvelopedEvent<AppRequestedEvent | ...>

  console.log('MES', message, body)

  if (message.files) {

    await client.chat.postMessage( options: {
      channel: 'U02033Y21NH',
      text: `Check this markup from <@${message.user}>: ${message.files[0].url_private}`,
    })

    answer = new RegExp( pattern: `${message.user}`, flags: 'i');
  }

});
```

Рис.4.5 – Функція відправлення графічного зображення

Slack Bolt запускається на сервері, відповідно, вся логіка від чатів і користувачів обслуговується в одному потоці, що кардинально відрізняється від стандартних React-додатків, які запускаються на стороні кожного клієнта.

Ще однією перевагою застосування Slack Bolt є той факт, що він не прив'язаний до жодного месенджеру. Існує основний пакет @slack/bolt, який дозволяє створювати абстрактних чат-ботів.

Для розв’язання проблеми порівняння графічних зображень між собою було обрано використання бібліотеки OpenCV.

OpenCV (Open Source Computer Vision Library) [26] – це бібліотека комп'ютерного зору з відкритим вихідним програмним кодом. В цій бібліотеці зібрана велика кількість алгоритмів для використання технологій комп'ютерного зору. Після підключення бібліотеки до свого проекту користувач отримує доступ до більш ніж 500 функцій, призначених для вирішення різноманітних завдань. Крім алгоритмів для роботи з технологіями комп'ютерного зору бібліотека також застосовується і для обробки зображень, містить велику кількість обчислювальних алгоритмів і багато іншого. Бібліотеку реалізовано мовами програмування C/C++, разом з тим вона поставляється також і для інших мов програмування.

Головною задачею бібліотеки OpenCV є надання легкого у використанні інтерфейсу, який допоможе полегшити використання технологій комп'ютерного зору в досить складних додатках. Функції, які підтримує бібліотека, охоплюють різноманітні сфери комп'ютерного зору – від медицини, безпеки до стереозображень та робототехніки.

Бібліотека OpenCV є структурованою бібліотекою. Вона розділена на п'ять основних компонентів: алгоритми обробки зображень, високорівневі алгоритми комп'ютерного зору, MLL – бібліотека машинного навчання, HighGUI – процедури та функції вводу-виводу для зберігання і завантаження відео і зображень, CXCore – основні структури даних.

До складу бібліотеки входить необхідний для розв’язання задачі дослідження детектор кутів Харріса.

Виклик детектора Харріса виконується за допомогою функції:

void cornerHarris(InputArray src, OutputArray dst, int blockSize, int ksize, double k, int borderType=BORDER_DEFAULT),

в якій параметри blockSize та k визначають кількість особливих точок на зображенні.

Для створення моделі заднього фону в бібліотеці OpenCV застосовують функцію віднімання кадрів:

absdiff(frameTime1, frameTime2, frameForeground),

де frameTime1 – перший кадр, frameTime2 – наступний кадр, frameForeground – результат виконання функції. Послідовно застосовуючи функцію детектора Харріса та віднімання фону на базовому скриншоті та на тих, що зберігаються в каталозі проекту, ми отримуємо можливість динамічно контролювати зміни, що відбуваються в дизайні інтерфесів та інформувати UI дизайнера про необхідність перевірити внесені зміни.

4.3 Програмна реалізація чат-бота

Прототип чат-боту складається з трьох основних частин – сценарій Docker HUB, який виконується як частина безперервного конвеєру для перевірки наявності змін в скриншотах [27], чат-додаток Bot та один або декілька зовнішніх інтерфейсів, в яких представлені дані (наприклад, Slack) (див. рис.4.6).

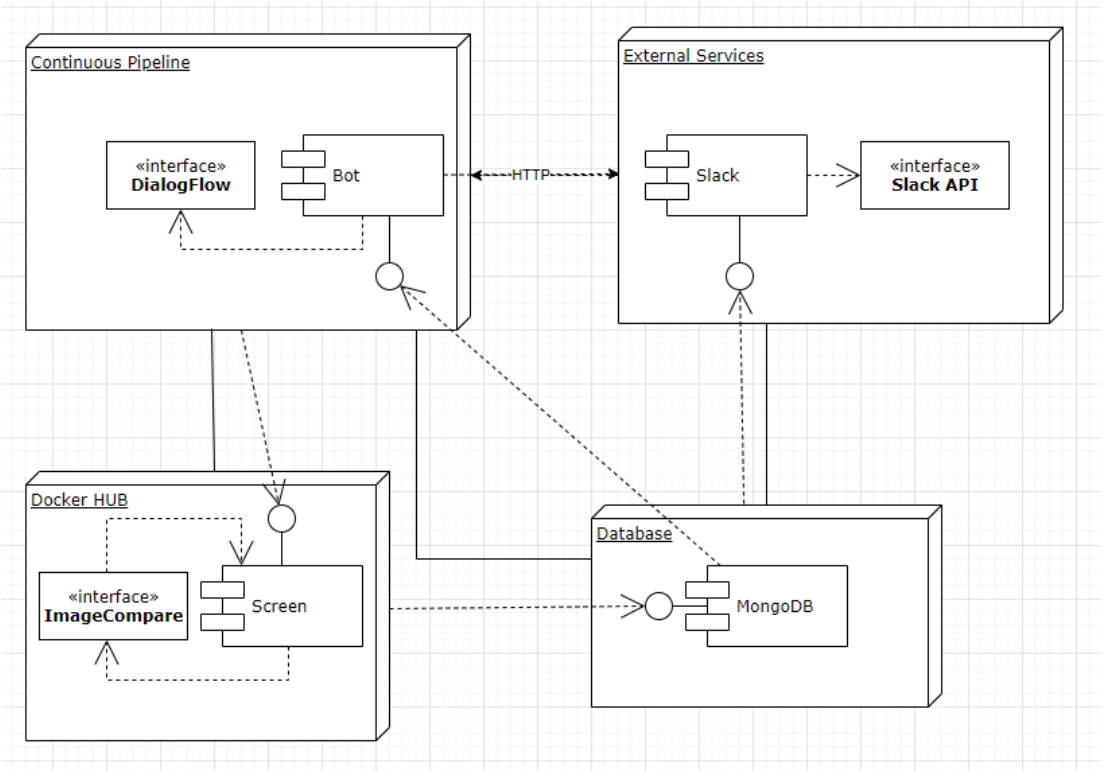

Рис.4.6 – Діаграма розгортання

З діаграми послідовностей (рис.4.7) бачимо, що безперервний конвеєр Continuous Pipeline постійно звертається до Docker HUB, очікуючи на настання події, коли результат порівняння останнього скріншоту з шаблоном, що зберігається в БД, будуть мати розбіжності (рис.4.7).

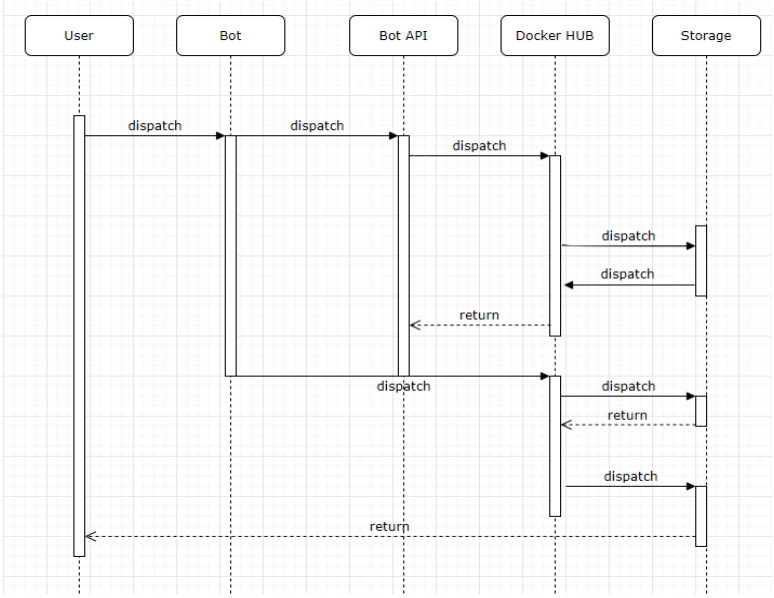

Рис. 4.7 – Діаграма послідовностей функціоналу чат-бота

Алгоритм порівняння скріншотів наведено на рис.4.8. Порівняння проводиться між тим скріншотом, що зберігається в БД, та останнім завантаженим у каталог проекту. Для збереження скріншотів використовуємо базу даних MongoDB – об'єктно-реляційну СКБД з відкритим вихідних кодом.

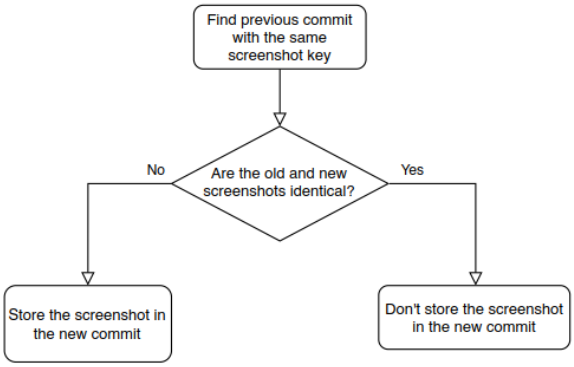

Рис.4.8 – Алгоритм процедури порівняння скриншотів

Для розробки чат-бота було виконано WebStorm (рис.4.9). WebStorm – це середовище розробки JavaScript-проектів, яке підходить як для розробки front-end, так і для створення додатків на Node.js. Головною перевагою WebStorm є зручний і розумний редактор JavaScript, HTML та CSS, до якого включено підтримку TypeScript, CoffeeScript, Dart, Less і Stylus і фреймворки – AngularJS, React і Meteor.

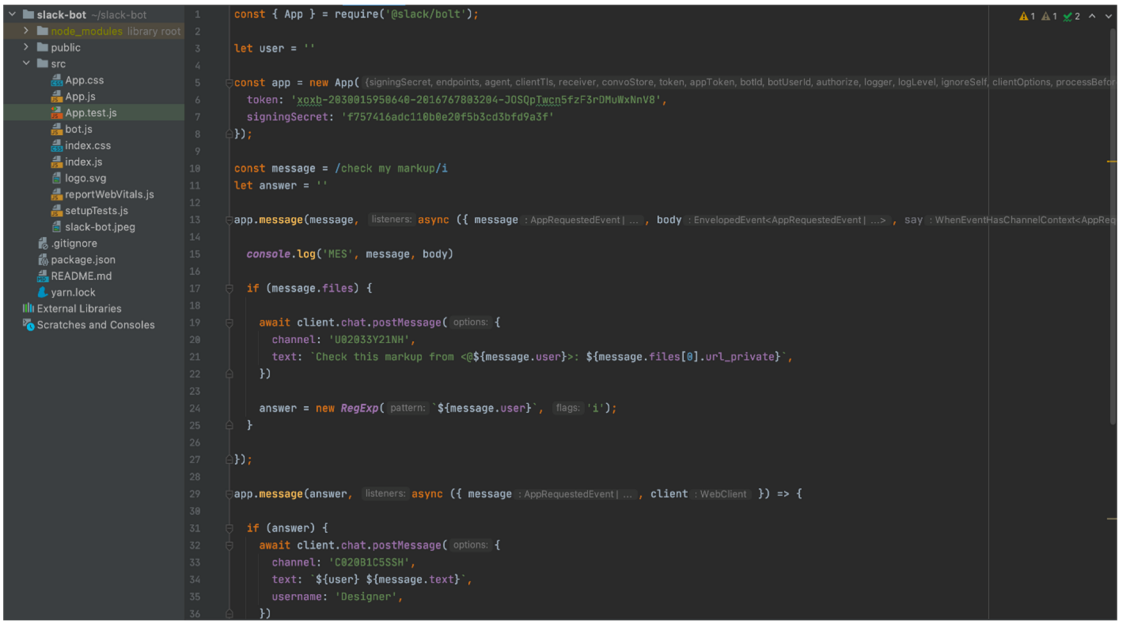

Рис.4.9 – Програмна реалізація

Для перевірки працездатності розробленого програмного забезпечення на GitHub було використано тестовий проект веб-сайту (https://github.com/website-template/html5-simple-personal-website), в якому результати змін дизайну веб-сторінок зберігались в каталозі Screenshots.

Робота з чат-ботом починається із його завантаження в месенджері Slack (рис.4.10).

```
Yegors-MacBook-Pro:slack-bot yegor$ yarn dev
yarn run v1.22.10
$ concurrently --kill-others "npm run client" "npm run bot"
[1]
[1] > slack-bot@0.1.0 bot
[1] > node src/bot.js
[1]
[0]
[0] > slack-bot@0.1.0 client
[0] > react-scripts start
[0]
[1] ⚡ Slack bot is running!
[0] ℹ ｢wds｣: Project is running at http://192.168.31.10/
```

Рис.4.10 – Запуск чат-бота в Slack

На сайті Slack API для відстеження роботи чат-бота через бібліотеку Slack Bolt можна отримати відповідь на віддалений запит, для чого надається посилання на відповідний event (рис.4.11).

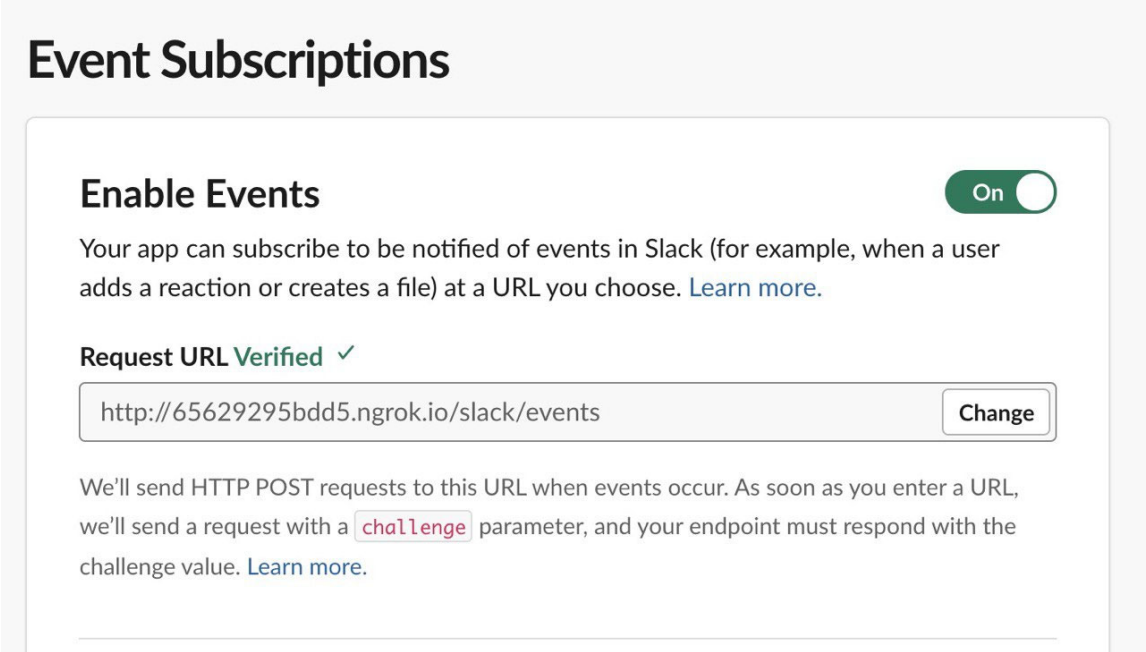

Рис.4.11 – Доступ до events через Slack

Для забезпечення функціоналу системи через протоколи http та https застосовується бібліотека ngrok (рис.4.12). Чат-бот готовий до використання.

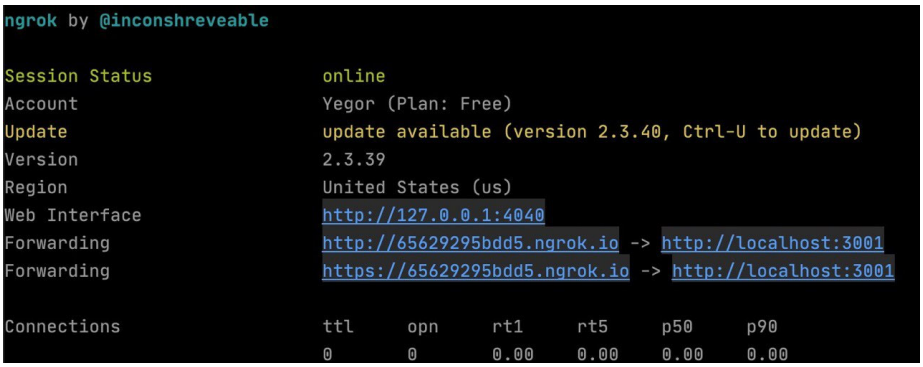

Рис.4.12 – Статистика використання бібліотеки ngrok

Після завантаження чат-боту дизайнеру необхідно завантажити до БД скріншот, який у подальшому буде використовуватись у якості шаблону (рис. 4.13).

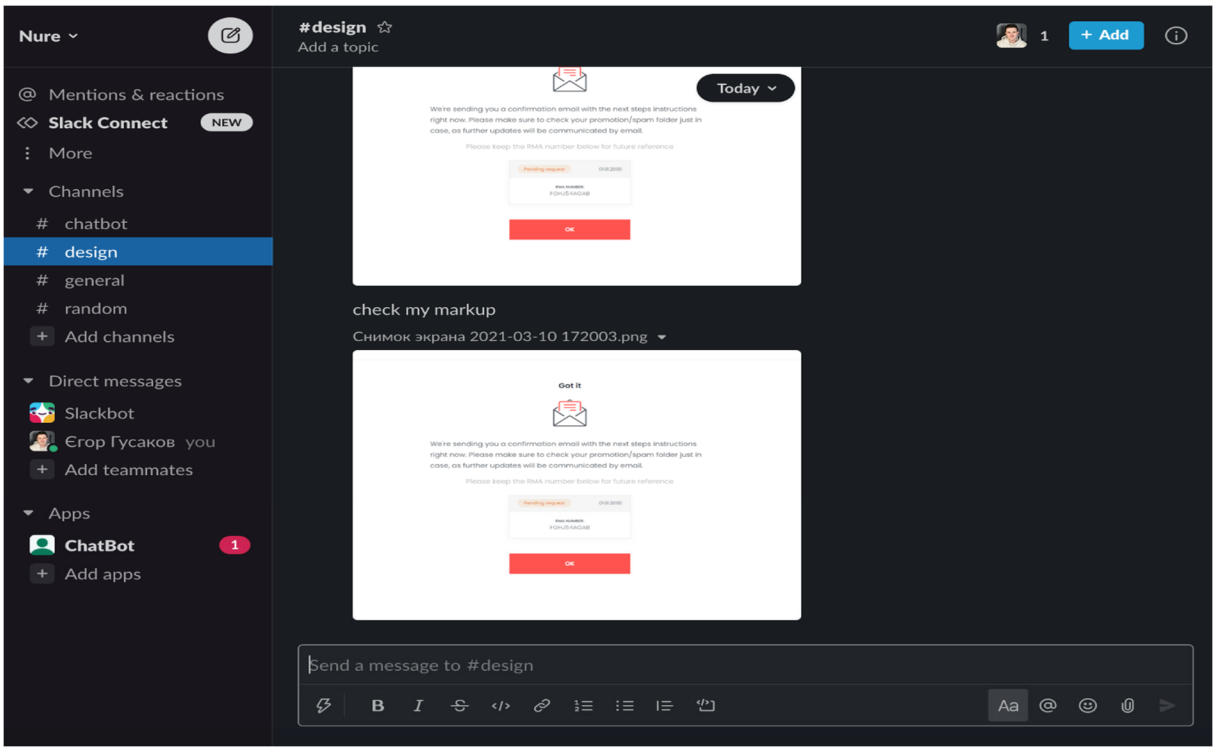

Рис.4.13 – Початок роботи з чат-ботом

Поки в дизайні веб-сторінки нічого не змінюється бот чекає на настання події в контейнері Docker HUB, який порівнює зображення в БД зі скріншотом з робочої директорії проекту.

Після настання події @different скрипт в контейнері Docker HUB надсилає до чат-бота нове зображення, що відрізняється від шаблону. Чат-бот, у свою чергу, надсилає це зображення у відповідний канал месенджера Slack (рис.4.12).

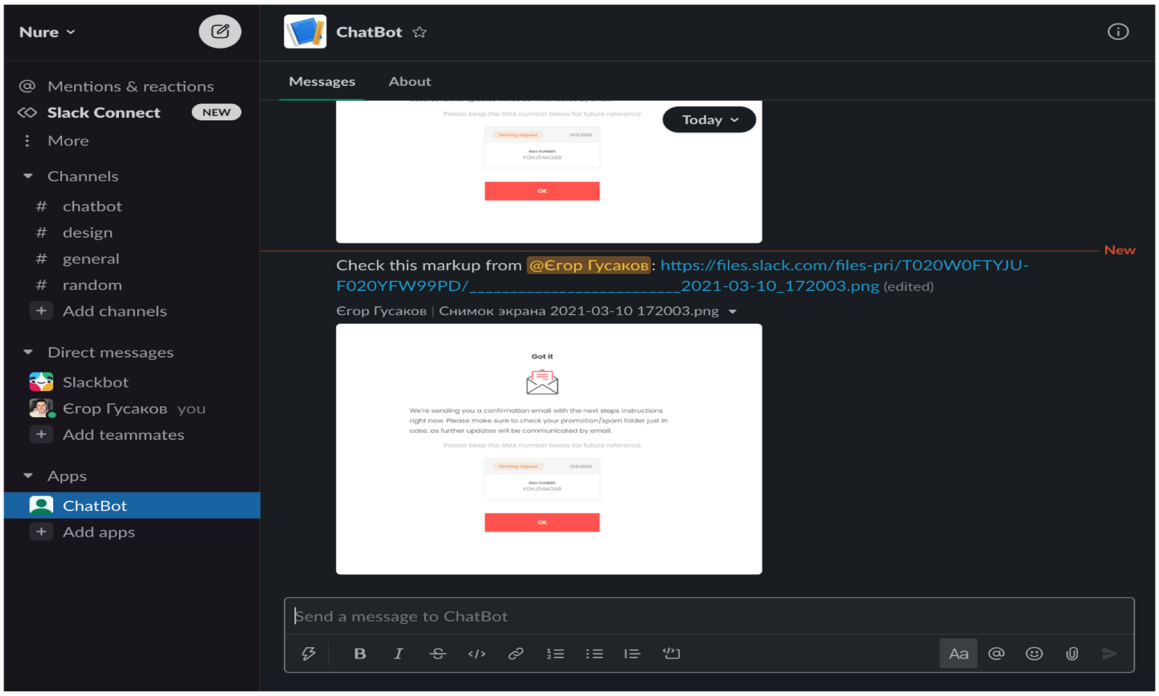

Рис.4.12 – Відповідь чат-боту у разі наявності відмінностей

У разі затвердження змін дизайнер надсилає до чат-боту команду «Perfect!», якою здійснюється заміна минулого шаблону скріншоту на останній, який у подальшому буде використовуватись у якості шаблону для продовження роботи над проектом (див. рис.4.13).

Якщо ж дизайнер не задовільнений результатом кодування інтерфейсу, то він виконує комунікацію з розробником звичним для себе способом для узгодження внесення змін у проект.

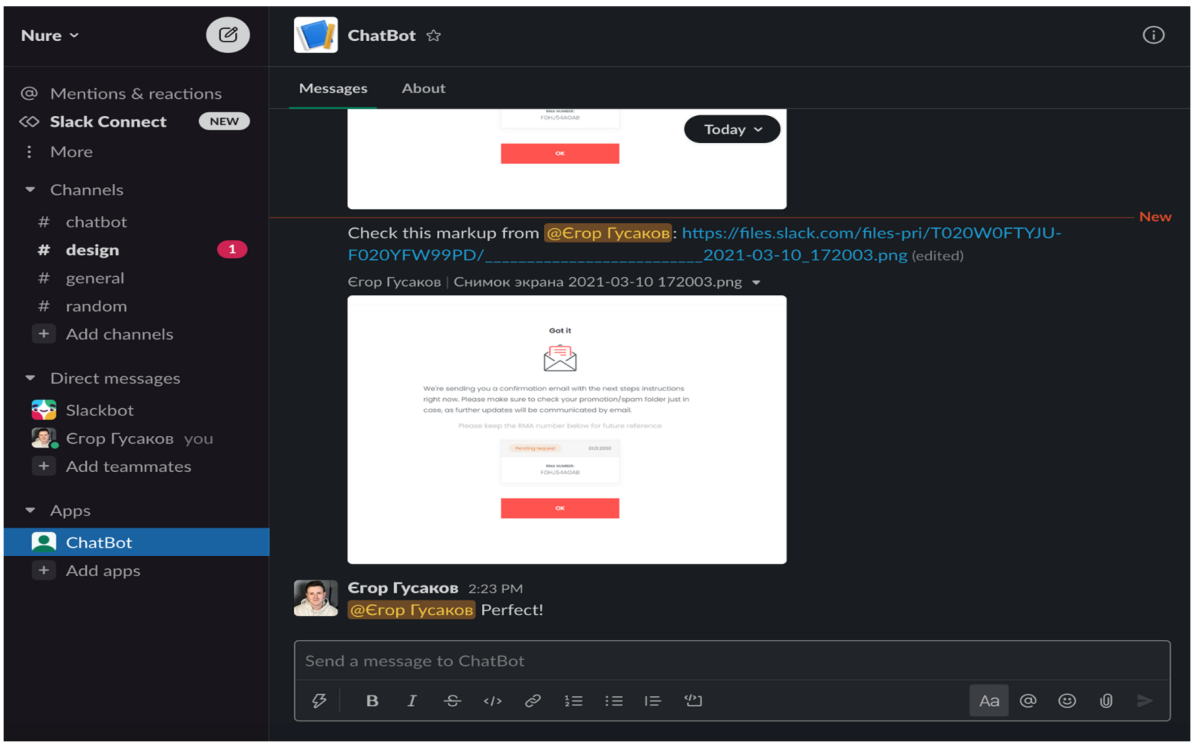

Рис.4.12 – Погодження дизайнером змін через чат-бот

У підсумку слід зазначити, що чат-боти є одним з найбільш перспективних нововведень для систем комунікації. Саме тому результати дослідження та запропоновані програмні рішення можуть бути використані для підвищення продуктивності команд розробників, покращення комунікативних можливостей, а подальше доопрацювання запропонованого прототипу програмної системи у вигляді чат-боту для месенджера Slack дозволить зменшити трудовитрати на проектування, прототипування, реалізацію та адаптацію користувацького інтерфейсу.

ВИСНОВКИ

В рамках написання кваліфікаційної роботи магістра було виконано наступне:

– проведено детальний аналіз предметної області дослідження, визначено особливості розробки інтерфейсів ПЗ за технологіями швидкої розробки;

– проаналізовано умови та засоби забезпечення комунікації між членами колективів розробників ПЗ та визначено проблеми, які впливають на якість роботи дизайнерів UI та їх співпрацю з розробниками коду ПЗ;

– виконано постановку задачі дослідження;

– спроектовано прототип програмної системи у вигляді чат-боту, який орієнтований на підвищення ефективності комунікаційної взаємодії між UI дизайнерами та розробниками;

– обрано засоби програмної реалізації програмної системи, яку у подальшому інтегровано в корпоративний месенджер;

– виконано перевірку працездатності програмної системи.

Як підсумок дослідження, проведеного в кваліфікаційної роботи, було представлено чат-бот для корпоративного месенджера Slack, який дозволяє покращити якість комунікації між окремим членами команди розробників ПЗ за рахунок випереджуючого інформування про зміни в дизайні, які відбуваються за рахунок внесення змін в програмний код. Застосування метода особливих точок для оцінювання ступеня змін в дизайні та можливість динамічної зміни шаблонів порівняння за рахунок використання контейнерної обробки зображень в нескінченному циклі роботи чат-бота дозволить спростити його використання на інші процеси розробки, а у подальшому дозволить створити повноцінну діалогову систему контролю за дотриманням встановлених вимог в процесі розробки ПЗ за технологіями швидкої розробки.

Результати дослідження, проведеного в рамках написання кваліфікаційної роботи, були представлені на міжнародній конференції «Interaction of society and science: prospects and problems» в квітні 2021 р. [28].

ПЕРЕЛІК ПОСИЛАНЬ

1. Бебик В. М. Інформаційно-комунікаційний менеджмент у глобальному суспільстві: психологія, технології, техніка паблік рилейшнз : монографія / В. М. Бебик. – Київ : МАУП, 2005. – 440 с.

2. Швайка Л. А. Планування діяльності підприємства: навч. посібник / Л. А. Швайка. – 4-те вид., стереотип. – Львів : Новий світ-2000, 2010. – 268 с.

3. Т. М. Бурмака. Комунікативний менеджмент: конспект лекцій / Т. М. Бурмака, К. О. Великих ; Харків. нац. ун-т міськ. госп-ва ім. О. М. Бекетова. – Харків : ХНУМГ ім. О. М. Бекетова, 2019. – 69 с.

4. О. Маркелов. Класифікація інтерактивних взаємодій користувача з програмним забезпеченням. // Вісник Національного університету “Львівська політехніка” “Комп’ютерні системи проектування. № 711, 2011, с. 15-21. URL: http://science.lpnu.ua/uk/cds-archive/vsi-vypusky/nomer-711-2011/klasyfikaciya-interaktyvnyh-vzayemodiy-korystuvacha-z (дата звернення: 15.02.2021).

5. Ben Nadel (2020). The most common mistake engineers make during the designer-developer handoff. URL: https://www.invisionapp .com/inside-design/developer-engineer-handoff-mistakes/?fbclid= IwAR2FAePSy0bXZkS3eQMdQLKUys1LwxDsuI8gPw45bqXZJDLwZKfll_unVz8 (дата звернення: 15.02.2021).

6. Maudet, Nolwenn et al. (2019). “Enact: Reducing designer–developer breakdowns when prototyping custom interactions”. In: ACM Transactions on Computer-Human Interaction (TOCHI) 26.3, pp. 1–48. URL: https://dl.acm.org/doi/10.1145/3310276 (дата звернення: 15.02.2021).

7. Akiki, Pierre et al. “Adaptive model-driven user interface development systems”. In: ACM Computing Surveys (CSUR) 47.1, pp. 1–33.

8. Rivero, Jose Matias et al. “Mockup-Driven Development: Providing agile support for Model-Driven Web Engineering”. In: Information and Software

Technology 56.6, pp. 670–687. URL: http://www.sciencedirect.com/science/article/pii/S0950584914000226 (дата звернення: 15.02.2021).

9. Cockton, Gilbert et al. Integrating User-Centred Design in Agile Development. isbn: 978-3-319-32163-9. doi: 10.1007/978-3-319-32165-3.

10. Jones, Alexander et al. “Collaboration Contrainsts for Designers and Developers in an Agile Environment”. URL: https://www.researchgate.net/publication/310624270_Collaboration_Contrainsts_for_Designers_and_Developers_in_an_Agile_Environment (дата звернення: 15.02.2021).

11. Ishii H., Ullmer B. Tangible bits: towards seamless interfaces between people, bits and atoms // Proceedings of the ACM SIGCHI Conference on Human factors in computing systems. ACM, 1997. P. 234–241. URL: https://www.researchgate.net/publication/2311166_Tangible_Bits_Towards_Seamless_Interfaces_between_People_Bits_and_Atoms (дата звернення: 18.02.2021).

12. Ratti C. et al. Tangible User Interfaces (TUIs): a novel paradigm for GIS // Transactions in GIS. 2004. Т. 8, № 4. P. 407–421. URL: https://www.media.mit.edu/publications/tangible-user-interfaces-tuis-a-novel-paradigm-for-gis-2/ (дата звернення: 18.02.2021).

13. Fishkin K.P., Moran T.P., Harrison B.L. Embodied user interfaces: Towards invisible user interfaces // Engineering for Human-Computer Interaction. Springer US, 1999. P. 1–18. URL: http://teilab-static.arch.tamu.edu/quek/Classes/Aware+EmbodiedInteraction/EmbodiedInteractionPAPERS/FisMH98.pdf (дата звернення: 19.02.2021).

14. User Interface Design Guidelines: 10 Rules of Thumb // Електронний ресурс: назва з екрану. URL: https://www.interaction-design.org/literature/article/user-interface-design-guidelines-10-rules-of-thumb?fbclid=IwAR1AYaIrkc1zFYsez5JwntDlSEliPBu_gv-PJZrTjDjYWqIY4vclM4XJySE (дата звернення: 19.03.2021).

15. Демарко, Т. Человеческий фактор: успешные проекты и команды / Т. Демарко, Т. Листер. – СПб.: Символ-Плюс, 2014. – 288 с.

16. Коул, Р. Блистательный Agile: гибкое управление проектами с помощью Agile, Scrum и Kanban [Електронний ресурс] / Р. Коул, Э. Скотчер; пер. с англ. – СПб.: Питер, 2019. – URL: http://flibusta.is/b/538856/read (дата звернення: 18.03.2021).

17. The Ultimate Guide to Employee Rewards & Recognition. // Електронний ресурс: назва з екрану. URL: https://blog.vantagecircle.com/cross-functional-teams/ (дата звернення: 18.03.2021).

18. Як перестати сидіти в чаті та почати працювати. 10 практик асинхронної комунікації. // Електронний ресурс: назва з екрану. URL: https://dou.ua/lenta/articles/practices-of-asynchronous-communication/ (дата звернення: 18.03.2021).

19. Як зробити корпоративний Slack швидким і продуктивним. // Електронний ресурс: назва з екрану. URL: https://dou.ua/lenta/articles/how-to-use-slack/ (дата звернення: 18.03.2021).

20. Построение SIFT дескрипторов и задача сопоставления изображений // Електронний ресурс: назва з екрану. URL:https://habrahabr.ru/post/106302/ (дата звернення: 13.04.2021).

21. Tuytelaars, T. Local invariant feature detectors: a survey / T. Tuytelaars, R. Mikolajczyk // Foundations and trends in computer graphics and vision - 2008. - Vol. 3(3). - P. 177-280. http://dx.doi.org/10.1561/0600000017 (дата звернення: 13.04.2021).

22. Smelyakov K., Chupryna A., Kolisnyk M., Ponomarenko O. Search by Image Engine Using Local Feature Detectors // 2020 IEEE Open Conference of Electrical, Electronic and Information Sciences (eStream), 30 April 2020, Vilnius, Lithuania. – P. 1-4. DOI: 10.1109/eStream50540.2020.9108884 (дата звернення: 16.04.2021).

23. Хабр. Нахождение объектов на картинках // Електронний ресурс: назва з екрану. URL: https://habr.com/ru/company/joom/blog/445354/ (дата звернення 17.04.2021).

24. Современный учебник JavaScript // Електронний ресурс: назва з екрану. URL: https://learn.javascript.ru/ (дата звернення 17.04.2021).

25. Slack Bolt // Електронний ресурс: назва з екрану. URL: https://www.npmjs.com/package/@slack/bolt (дата звернення 17.04.2021).

26. OpenCV. Open Source Computer Vision Library // Електронний ресурс: назва з екрану. URL: http://SourceForge.net/projects/opencvlibrary (дата звернення 17.04.2021).

27. Система уведомлений о событиях (Webhooks). // Електронний ресурс: назва з екрану. URL: https://www.unisender.com/ru/support/api/common/sistema-uvedomlenij-o-sobytiyah-webhooks/ (дата звернення 22.04.2021).

28. Гусаков Є. Використання чат-ботів для комунікацій між UI дизайнерами та розробниками / The XXII International Science Conference «Interaction of society and science: prospects and problems», April 20 – 23, 2021, London, England. P. 560-562.