БАКАЛАВРСЬКА РОБОТА

БР. ІП - 23.00.00.000 ПЗ

Стефанишин Олег

2025

Івано-Франківський національний технічний університет нафти і газу
Інститут інформаційних технологій
Кафедра інженерії програмного забезпечення

Стефанишин Олег Ігорович

(прізвище, ім'я, по батькові)

УДК 004.942
(індекс)

БАКАЛАВРСЬКА РОБОТА

Програмні технології проектування інтерфейсу користувача

(назва роботи)

Інженерія програмного забезпечення

(назва освітньої програми)

121– Інженерія програмного забезпечення

(шифр і назва спеціальності)

Робота містить результати власних досліджень, використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело:

Здобувач освітнього ступеня _____Стефанишин О. І._____
(підпис, ініціали та прізвище здобувача)

Науковий керівник ____Процюк Володимир Васильович, к.т.н., ____
(підпис, прізвище, ім'я, по батькові, науковий ступінь, вчене звання керівника)

Допущено до захисту
Завідувач кафедри

_доц._____Бандура В.В.__
(посада) (підпис) (дата) (ініціали та прізвище)

Івано-Франківськ – 2025

Івано-Франківський національний технічний університет нафти і газу

Інститут, факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Ступінь вищої освіти бакалавр

Спеціальність 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
Зав. кафедрою **ІПЗ**
доцент. В.В. Бандура
“ ____ ” _____2025 р.

ЗАВДАННЯ
НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТОВІ

Стефанишин Олег Ігорович

(прізвище, ім’я, по-батькові)

1. Тема проекту (роботи) "Програмні технології проектування інтерфейсу користувача”

керівник проекту (роботи) к.т.н., . Процюк Володимир Васильович

затверджені наказом вищого навчального закладу від “ 28 ” квітня 2025 р. № 264/7

2. Строк подання студентом проекту (роботи) 10 червня 2025 р.

3. Вихідні дані до проекту (роботи) Результати і матеріали отримані під час проходження переддипломної практики

4. Зміст розрахунково - пояснювальної записки(перелік питань, які потрібно розробити)

1. Концептуальні основи проєктування інтерфейсу користувача
2. Огляд інструментів проєктування інтерфейсу користувача
3. Інструменти та підходи до проєктування інтерфейсу користувача

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

1. Модель експертності користувачів у порівнянні зі складністю медичних інтерфейсів користувача (рис.1.1, ст. 28)
2. Різні організації, які використовували віконні системи 41(рис.2.2, ст. 41)
3. Екран з оригінального Macintosh, на якому показано 3 вікна, що накривають одне одного, та деякі піктограми вздовж правого поля (рис.2.3, ст. 46)
4. Компоненти інтерфейсу користувача програмного забезпечення (рис.3.1, ст. 56)
5. Потік подій (рис.3.3, ст. 65)

6. Консультанти розділів проекту (роботи)

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

2. Дата видачі завдання **10 травня 2025 р.**

Керівник _____
(підпис)

Завдання прийняв до виконання_____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту	Примітка
1	Визначення та обґрунтування теми роботи	15.02.2025	виконано
2	Огляд існуючих концепцій, рішень та сервісів в даній області	25.02.2025	виконано
3	Побудова моделі або алгоритму власного рішення	15.03.2025	виконано
4	Документування реалізації власного оригінального рішення вибраними засобами	25.04.2025	виконано
5	Оформлення пояснювальної записки бакалаврської роботи	10.06.2025	виконано

Студент _____
(підпис)

Керівник роботи _____
(підпис)

АНОТАЦІЯ

Бакалаврська робота містить 70 сторінок, 11 рисунків, список використаних джерел із 27 найменування,

Метою роботи є проаналізувати сучасні концепції, інструменти та підходи до проектування інтерфейсу користувача з метою виявлення ефективних рішень для створення зручних, адаптивних та функціональних користувачів.

Об'єкт дослідження: процес проектування інтерфейсу користувача в інформаційних системах.

Предмет дослідження: програмні інструменти, технології та підходи, що використовують для проектування інтерфейсу користувача, а також їх вплив на ефективність, зручність та адаптивність коду.

Результати дослідження: було встановлено, що ефективне проектування інтерфейсу користувача залежить від поєднання еволюційно сформованих підходів.

У першому розділі розглядаються концептуальні основи проєктування інтерфейсу користувача, включаючи еволюцію підходів, моделі експертності користувачів і вплив досвіду на оцінку зручності використання.

Другий розділ присвячений огляду інструментів проєктування UI, з фокусом на їх важливість, огляд програмного забезпечення та набори інструментів.

Третій розділ зосереджується на сучасних інструментах і підходах, таких як інструменти вищого рівня, автоматична генерація на основі моделей і технології для веб-розробки.

Висновок: узагальнено ключові результати та окреслено перспективи розвитку програмних технологій проєктування UI**.**

КЛЮЧОВІ СЛОВА: ІНТЕРФЕЙС КОРИСТУВАЧА, ПРОГРАМНІ ТЕХНОЛОГІЇ, ПРОЄКТУВАННЯ UI, FIGMA, REACT, ПРОТОТИПУВАННЯ, ЗРУЧНІСТЬ ВИКОРИСТАННЯ, АДАПТИВНИЙ ДИЗАЙН.

ANNOTATION

The bachelor's thesis contains 70 pages, 11 figures, a list of used sources with 27 names,

The purpose of the work is to analyze modern concepts, tools and approaches to user interface design in order to identify effective solutions for creating convenient, adaptive and functional users.

Object of research: the process of user interface design in information systems.

Subject of research: software tools, technologies and approaches used for user interface design, as well as their impact on the efficiency, usability and adaptability of the code.

Research results: it was found that effective user interface design depends on a combination of evolutionarily formed approaches.

The first section examines the conceptual foundations of user interface design, including the evolution of approaches, user expertise models, and the impact of experience on usability evaluation.

The second section provides an overview of UI design tools, focusing on their importance, software overview, and toolkits.

The third section focuses on modern tools and approaches, such as higher-level tools, automatic model-based generation, and web development technologies**.**

Conclusion: Key findings are summarized and the future of software-based UI design technologies is outlined.

KEYWORDS: USER INTERFACE, SOFTWARE TECHNOLOGIES, UI DESIGN, FIGMA, REACT, PROTOTYPING, USABILITY, ADAPTIVE DESIGN.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

НСІ - Людино-комп'ютерна взаємодія

ЧКІ - аспектів людської кон'юнктури

UCD - орієнтований на користувача

HMI - взаємодії людини з машиною

CAH - теорія гіпотези налаштування обмежень

ТКГ - Теорію когнітивної гнучкості

API - інтерфейсом програмування додатків

EET - Elements-Events and Transitions

IDT - інструментами розробки інтерфейсів

UIL - мова інтерфейсу користувача

4GL - Мови четвертого покоління

WAP - протокол бездротового доступу

UIMS - Система керування інтерфейсом користувача

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ……8

ВСТУП 9

РОЗДІЛ 1. КОНЦЕПТУАЛЬНІ ОСНОВИ ПРОЄКТУВАННЯ ІНТЕРФЕЙСУ КОРИСТУВАЧА 11

1.1. Еволюція підходів до проєктування інтерфейсів 11

1.2. Модель експертності користувачів проти складності інтерфейсів користувача 27

1.3. Вплив досвіду користувача на оцінку зручності використання 29

1.4 Висновок до розділу……..…..34

РОЗДІЛ 2. ОГЛЯД ІНСТРУМЕНТІВ ПРОЄКТУВАННЯ ІНТЕРФЕЙСУ КОРИСТУВАЧА 34

2.1. Важливість інструментів інтерфейсу користувача 34

2.2. Огляд інструментів програмного забезпечення для інтерфейсу користувача 35

2.3. Набори інструментів 47

2.4 Висновок до розділу……53

РОЗДІЛ 3. ІНСТРУМЕНТИ ТА ПІДХОДИ ДО ПРОЄКТУВАННЯ ІНТЕРФЕЙСУ КОРИСТУВАЧА 54

3.1. Інструменти вищого рівня 54

3.2. Автоматична генерація на основі моделі 63

3.3. Інструменти для Всесвітньої павутини 74

3.4 Висновок до розділу……78

ВИСНОВКИ 79

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ……81

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

HCI - Людино-комп'ютерна взаємодія

ЧКІ - аспектів людської кон'юнктури

UCD - орієнтований на користувача

HMI - взаємодії людини з машиною

CAH - теорія гіпотези налаштування обмежень

ТКГ - Теорію когнітивної гнучкості

API - інтерфейсом програмування додатків

EET - Elements-Events and Transitions

IDT - інструментами розробки інтерфейсів

UIL - мова інтерфейсу користувача

4GL - Мови четвертого покоління

WAP - протокол бездротового доступу

UIMS - Система керування інтерфейсом користувача

ВСТУП

Дипломна робота присвячена дослідженню програмних технологій проєктування інтерфейсу користувача (UI) як інструменту для створення інтуїтивно зрозумілих, ефективних і адаптивних інтерфейсів у цифрових продуктах. Метою роботи є аналіз сучасних програмних підходів і інструментів для проєктування UI, їх еволюції та практичного застосування для підвищення зручності використання в різних галузях, включаючи веб-розробку, мобільні додатки та корпоративні системи. Дослідження зосереджується на теоретичних основах проєктування, програмних інструментах і викликах, пов’язаних із створенням користувацько-орієнтованих інтерфейсів.

Проаналізовано основні програмні інструменти, такі як Figma, Sketch, Adobe XD, а також бібліотеки та фреймворки (React, Angular, Vue.js), з точки зору їх функціональності, можливостей прототипування та інтеграції з розробкою. Окремо досліджено сучасні підходи, такі як автоматична генерація UI на основі моделей і інструменти для веб-розробки.

Практична частина дослідження включає розробку прототипу інтерфейсу користувача для конкретного сценарію, наприклад, створення адаптивного веб-додатка. Оцінюються такі аспекти, як зручність використання інструментів, продуктивність прототипу, кросплатформна сумісність і відповідність стандартам доступності. Додатково проведено порівняльний аналіз інструментів проєктування за критеріями ефективності, гнучкості та підтримки спільноти.

Проєктування інтерфейсу користувача (UI) є ключовим елементом сучасного програмного забезпечення, яке визначає якість взаємодії між людиною та цифровими системами. У контексті швидкого розвитку технологій, програмні технології проєктування UI відіграють вирішальну роль у створенні інтуїтивно зрозумілих, ефективних і адаптивних інтерфейсів, що відповідають потребам різноманітних користувачів. Від традиційних десктопних додатків до веб-платформ і мобільних застосунків, програмні інструменти, такі як Figma, Sketch, Adobe XD, а також бібліотеки React, Angular і Vue.js, дозволяють

розробникам створювати інтерфейси, що поєднують естетику, функціональність і зручність використання. Ці технології не лише спрощують процес проєктування, але й забезпечують адаптацію до нових викликів, таких як кросплатформна сумісність і автоматизація розробки.

Актуальність теми зумовлена стрімким зростанням попиту на високоякісні інтерфейси в умовах цифрової трансформації. За даними аналітичних звітів, таких як Nielsen Norman Group, ефективний UI може значно підвищити задоволеність користувачів і продуктивність систем, тоді як погано спроєктовані інтерфейси призводять до зниження залученості та економічних втрат. Водночас сучасні виклики, такі як різноманітність пристроїв, складність користувацьких сценаріїв і потреба в автоматизації проєктування, підкреслюють важливість програмних технологій, які дозволяють оптимізувати розробку та підвищувати якість інтерфейсів.

Метою цього дослідження є аналіз сучасних програмних технологій проєктування інтерфейсу користувача, включаючи їх концептуальні основи, інструменти та підходи до створення ефективних UI. Робота спрямована на систематизацію знань про еволюцію методів проєктування, інструменти розробки та їх вплив на зручність використання, а також на надання практичних рекомендацій для розробників і дизайнерів. Дослідження охоплює як теоретичні аспекти, так і практичні інструменти, що застосовуються в сучасних UI-проєктах.

Цей документ має на меті надати комплексне уявлення про програмні технології проєктування інтерфейсу користувача, а також практичні рекомендації для фахівців, які прагнуть створювати інноваційні та зручні інтерфейси в умовах швидкозмінного технологічного середовища.

Завданнями дослідження: проаналізувати концептуальні основи сучасного проектування інтерфейсу користувача, когнітивні аспекти сприйняття інтерфейсів. Систематизувати методи та підходи, які застосовуються при створенні ефективних інтерфейсів користувача, огляд та класифікація сучасних інструментів UI/UX-дизайну та розробки, таких як Figma, Adobe XD, Sketch,

Axure, а також front-end фреймворків та бібліотек (React, Angular, Vue тощо, дослідити еволюцію технологій проектування UI, оцінити вплив обраних технологій та підходів на зручність використання, доступність, продуктивність та загальне враження користувача, провести огляд практичних кейсів застосування сучасних UI-технологій у реальних проектах, розробити практичні рекомендації для дизайнерів та розробників щодо вибору інструментів, технологій та методик для ефективного створення інтерфейсів.

Результати дослідження: було систематизовано концептуальні основи UI-дизайну, описано основні підходи до проектування інтерфейсів.

Об'єкт дослідження: процес проектування інтерфейсу користувача в інформаційних системах.

Предмет дослідження: програмні інструменти, технології та підходи, що використовують для проектування інтерфейсу користувача, а також їх вплив на ефективність, зручність та адаптивність коду**.**

Методи дослідження: аналіз літератури, порівняльний аналіз технологій, спостереження, опитування, тестування, моделювання симуляція.

Бакалаврська робота містить 60 сторінок, 15 рисунків, список використаних джерел із 20 найменуванням.

РОЗДІЛ 1 КОНЦЕПТУАЛЬНІ ОСНОВИ ПРОЄКТУВАННЯ ІНТЕРФЕЙСУ КОРИСТУВАЧА

1.1 Еволюція підходів до проєктування інтерфейсів

Людино-комп'ютерна взаємодія (HCI) – це міждисциплінарна галузь досліджень, яка стосується вивчення взаємодії між людьми (операторами як користувачами) та комп'ютерами. Широко вживане визначення тлумачить це поняття як «дисципліну, що займається проектуванням, оцінкою та впровадженням інтерактивних обчислювальних систем для використання людиною та вивченням основних явищ, що їх оточують» (ACM SIGCHI, 1992). Причиною визначення «міждисциплінарної» природи HCI є залучення різних дисциплін, що сприяють HCI. Інформатика – не єдина дисципліна, що сприяє HCI; інші дисципліни, такі як когнітивна психологія, людський фактор, інженерія, дизайн, соціальна та організаційна психологія, також вважаються важливими та актуальними.

Протягом багатьох років було проведено або впроваджено широкий спектр досліджень та досліджень, що стосуються різних аспектів людської кон'юнктури (ЧКІ). Ці дослідження різноманітні та включають, наприклад, методи ведення діалогу, жестовий аналіз та мультимодальні інтерфейси, комп'ютерну графіку, комп'ютерну лінгвістику, просторове пізнання, навігацію та орієнтування роботів, стилі або пристрої введення, екрани моніторів тощо. Однак кінцевою метою досліджень є сприяння покращенню взаємодії між людьми та комп'ютерними системами шляхом підвищення зручності використання та задоволення від використання технічних систем. Сприяючи взаємній взаємодії шляхом представлення інформації про стан комп'ютерних систем, інтерфейс користувача зазвичай працює як своєрідна комунікаційна платформа або міст між людьми та комп'ютерами під час взаємодії. Однак люди працюють як користувачі, які керують системою або контролюють її, обробляючи та інтерпретуючи інформацію. Як розробити інтерфейси, які

допомагають користувачам виконувати завдання оптимальним чином під час взаємодії, є головним завданням для всіх інженерів-конструкторів. Коли користувачі взаємодіють з комп'ютерними системами, багато факторів можуть впливати на загальну продуктивність. Ці фактори охоплюють різні питання з боку користувача, з боку завдання, з боку технічної системи та з боку робочого середовища/контексту. Нещодавно, з точки зору інженерії людських факторів, [3], запропонували фактори, що формують продуктивність (ФПП), як загальну назву для цих різних впливових факторів. Крім того, вони класифікували ФПП на три категорії: внутрішні фактори, зовнішні фактори та стресори. Внутрішні фактори стосуються фізичних та психічних станів, які є або вбудованими, або привнесеними людьми як операторами, такими як вік, зір, особистість та мотивація тощо. Зовнішні фактори стосуються прихованих та операційних проблем, що стосуються робочого середовища та робочих контекстів, таких як навколишнє середовище, чергування змін, обладнання та інструменти, робоча процедура та інтерфейси оператора тощо. Стресори - це здебільшого психологічний або фізіологічний тиск, що впливає на прийняття рішень та дії оператора прямо чи опосередковано, такий як високе робоче навантаження, високий темп роботи, біль, виснаження, тривалий стрес тощо. Ці фактори, що формують продуктивність, впливають на продуктивність оператора окремо або в поєднанні. Метою узагальнення та вивчення різних впливових факторів є надання корисних висновків для комп'ютерних інженерів з метою оптимізації дизайну взаємодії.

Відповідно, в останні роки з'явилася низка різних методологій проектування, що розглядають методи проектування HCI. Обґрунтування більшості методологій проектування базується на моделі, що ілюструє взаємодію користувача, дизайнера та технічних систем. У перші роки когнітивні процеси користувачів вважалися важливими передбачуваними питаннями в багатьох методологіях проектування. Сьогодні хороша та ефективна комунікація між користувачами та дизайнерами розглядається як фокус у сучасних філософіях проектування, що спонукає технічні системи охоплювати той тип

досвіду, який хочуть мати користувачі. Дизайн, орієнтований на користувача (UCD), — це дуже популярна філософія проектування, метою якої є поставити користувачів у центр процесу проектування технічних систем, а також приділити значну увагу вимогам та обмеженням кінцевих користувачів на кожному етапі процесу проектування. Вимоги користувача розглядаються та розглядаються як фокус у всьому життєвому циклі продукту. Типовими характеристиками філософії дизайну UCD є активна участь кінцевих користувачів у процесі проектування та ітерація дизайнерських рішень. Тому дизайнери повинні не лише аналізувати або передбачати, як користувачі взаємодіють з інтерфейсом, але й перевіряти обґрунтованість своїх припущень щодо поведінки користувачів у реальних тестах з реальними користувачами. Порівняно з іншими філософіями дизайну інтерфейсів, UCD зосереджується на оптимізації інтерфейсу користувача навколо того, як користувачі можуть, хочуть або повинні працювати, а не змушує користувачів змінювати свої ментальні моделі чи поведінку, щоб пристосуватися до підходу дизайнерів. Для досягнення цієї гіпотези дизайну користувачі, дизайнери та технічні фахівці працюють разом, щоб сформулювати бажання, потреби та обмеження користувача та створити технічну систему, яка відповідає цим вимогам.

Основна ідея філософії UCD полягає в тому, щоб наголосити на хорошому розумінні користувача, щоб розробляти більш корисні артефакти. Різноманітність потреб передбачає врахування користувачів з різним досвідом, знаннями, віком, статтю, культурними відмінностями тощо (Schneiderman, 2000). Для того, щоб зрозуміти користувачів та їхні потреби, аналіз користувачів стає критичним аспектом процесу UCD. Аналіз користувачів часто означає широке розмежування користувачів за віком, статтю, досвідом роботи з технологіями, освітою, ставленням до технологій, лінгвістичними здібностями тощо.

Опис користувачів та робочих завдань є критично важливим для досліджень системи HCI. У загальноприйнятому сенсі, користувачі зазвичай характеризуються як клас людей, які використовують технічну систему та не обов'язково мають повний технічний досвід роботи з нею. Протягом досить

тривалого часу концепція користувача розмивалася різними школами мислення. Користувачі визначаються по-різному в різних теоріях: як компоненти системи в «Теорії розподіленого пізнання», як розв'язувачі проблем у «Теорії обробки інформації», як винахідливі особистості в «Теорії ситуативних дій», як людські актори в «Теорії діяльності» тощо. Однак з практичної точки зору, визначення користувачів завжди ґрунтується на їхньому ставленні до продуктів або артефактів. [1], зазначив, що визначення користувачів має бути пов'язане з використанням-діяльністю, тобто типом відношення людина-артефакт. Таким чином, користувач визначається як кінцевий користувач, тобто «людина, яка займається використанням-діяльністю» [1]. [2] визначив користувача як «будь-яку особу, яка з певною метою взаємодіє з продуктом або будь-яким реалізованим елементом (системою, деталлю, компонентом, модулем, функцією тощо, що проявився в програмному забезпеченні або як конкретні об'єкти) продукту на будь-якому етапі життєвого циклу продукту». Це визначення є більш повним та практичним з точки зору проектування та розробки продукту, оскільки враховується взаємодія людини з продуктом протягом усього життєвого циклу продукту.

Користувачів можна класифікувати за різними критеріями. Наприклад, багато дослідників класифікують користувачів за їх зв'язком з продуктами або іншими користувачами. [5], класифікував користувачів на цільові групи та групи фільтрів. Згідно з [5], користувачі в цільових групах – це особи, для яких розроблено або спроектовано продукт, тоді як користувачі в групах фільтрів можуть бути дистриб'юторами або покупцями, які можуть впливати на вибір продукту цільовими групами. [10] класифікували користувачів на дві категорії: первинні користувачі та вторинні користувачі. Первинні користувачі – це ті, хто використовує продукт за його основним призначенням, наприклад, медсестри з діалізу, які працюють з апаратом для діалізу під час медичного лікування, тоді як вторинні користувачі – це ті, хто активно використовує продукт, але не за його основним призначенням, наприклад, обслуговуючий персонал. На основі класифікації [9] додав ще дві категорії користувачів: побічні користувачі та

співкористувачі. Побічні користувачі – це «люди, на яких продукт впливає, негативно чи позитивно, у їхньому повсякденному житті, але вони не використовують цей продукт», наприклад, пацієнти, які проходять лікування за допомогою апарату штучної вентиляції легень, тоді як співкористувачі – це «люди, які певним чином співпрацюють з основним або вторинним користувачем, не використовуючи той самий продукт», наприклад, лікарі.

Однак, у HCI індивідуальні відмінності користувачів та їхні завдання завжди вказуються як два найважливіші питання, що стосуються зручності використання [15]. Користувачі не є однорідними; вони відрізняються за багатьма параметрами, такими як стать, вік, фізичні здібності, рівень освіти, організаційна культура, навички оператора тощо. Саме ці відмінності роблять користувачів різними один від одного як особистостей та подібними як колективну групу. Важливість характеристик користувача, наприклад, віку, статі, розмірів тіла та підготовки, завжди підкреслюється, оскільки характеристики користувача можуть впливати на ситуацію використання і, таким чином, впливати на дизайн продукту [20]. Враховуючи, що характеристики користувача визначають його здібності та обмеження в ситуації використання, багато дослідників класифікували користувачів на основі характеристик користувача та проводили дослідження, щоб дослідити, як ці характеристики впливають на продуктивність користувачів під час взаємодії з технічними системами. Наприклад, у літніх користувачів було виявлено зниження когнітивних процесів вищого порядку, таких як увага [12], а також нижчу швидкість виконання майже всіх завдань, що вимагають швидкої роботи [23], провів дослідження гендерних аспектів і виявив, що жінки демонструють вищу комп'ютерну тривожність, ніж чоловіки, що свідчить про те, що чоловіки більш впевнені в собі, а жінки більш нерішучі.

Мета класифікації користувачів полягає в ідентифікації або дослідженні користувачів та їхньої продуктивності чи прийняття дизайну продукту, тим самим надаючи корисну інформацію або основу для дизайнерів для вдосконалення продукту. [11] наголосили на важливості врахування досвіду

використання продукту користувачами [23] визначив три різні типи досвіду використання, які призводять до різних сприятливих ефектів у виявленні користувачів: «досвід вирішення проблем, що сприяє виникненню у користувачів, наприклад, проблем з існуючим дизайном продукту; досвід взаємодії, що сприяє виникненню у користувачів досвіду взаємодії з інтерфейсом користувача та усвідомлення властивостей продукту; та досвід використання продукту, що сприяє виникненню у користувачів досвіду використання продукту в діяльності користувача, тобто ситуації, в якій продукт став посередником для досягнення мети користувача». Однак, розглядаючи досвід використання продукту користувачами з точки зору оцінки продукту та взаємодії, досвід використання тут класифікується як досвід взаємодії та досвід використання. Досвід взаємодії стосується того, як користувачі отримують досвід використання продукту, взаємодіючи з інтерфейсом користувача в конкретній або певній ситуації, наприклад, у тестах зручності використання. Досвід використання стосується досвіду, який користувачі отримують, використовуючи продукт у реальному житті. [27] зазначив, що тривалість використання та знання щодо продукту, а також частота використання, є двома основами для визначення досвіду використання продукту користувачами. Однак, окрім тривалості використання та частоти використання, рівень знань користувачів є найпрямішим і найточнішим критерієм для визначення досвіду використання продукту користувачами.

Профілі користувачів

Як популярний термін, що широко використовується промисловими компаніями для представлення реальних користувачів, профіль користувача – це метод представлення даних з досліджень характеристик користувачів [24]. Він також може бути доповнений описом взаємозв'язків між різними користувачами. [6], зазначив, що «профіль користувача» майже те саме, що й персона, тобто певна вигадана особа як сукупність атрибутів (наприклад, цілі, вік, ставлення, робота та навички). Іншими словами, профіль користувача цільової групи містить колективну інформацію про психічні, фізичні та демографічні дані для

групи користувачів, а також інші характеристики. Можна створювати профілі користувачів для одного або кількох вигаданих окремих користувачів у формі персон, описуючи таким чином характеристики користувача у формі знань, здібностей та обмежень щодо продукту, обладнання чи системи, з якою користувач буде інтегруватися [22], Кінцева мета використання профілів користувачів або персон полягає в тому, щоб допомогти дизайнерам розпізнати або дізнатися про реального користувача, наприклад, представивши йому опис атрибутів реального користувача; стать, вік, рівень освіти, ставлення, технічні потреби та рівень навичок користувача. Профіль користувача не обов'язково відображає або представляє повний набір атрибутів усієї групи користувачів. Суть профілів користувачів полягає в точному та простому наборі атрибутів користувачів. У процесі проектування продукту профілі користувачів зазвичай створюються на ранніх стадіях і використовуються як основа для оцінки зручності використання та редизайну продукту.

Вік – це важливе питання, яке слід враховувати. Було виявлено, що старіння призводить до погіршення, наприклад, фізіології та нейрофізіології ока [21]; фізичних, сенсорних та когнітивних факторів [19]; когнітивних процесів вищого порядку, таких як увага [12], та до уповільнення майже всіх завдань, що вимагають швидкого виконання [14] Загальновизнано, що таке зниження прискорюється після досягнення людьми сорока п'яти років [10]. Водночас стверджується, що люди старшого віку стають більш обережними [9], і тому більш схильні планувати, перш ніж діяти, а не застосовувати метод спроб і помилок. У комп'ютерній роботі було виявлено, що люди старшого віку витрачали на це більше часу та робили більше помилок [2], а також демонстрували повільнішу продуктивність та допускали більше помилок у використанні пристроїв введення [2] провели дослідження, що порівнювало поведінку молодших та середнього віку під час взаємодії зі складними електронними пристроями. Їхні результати показали, що вікові відмінності суттєво впливали на спостережувану частоту помилок, кількість кроків взаємодії, жорсткість дослідження, успішність методів фізичних операцій та

суб'єктивне сприйняття часової потреби та продуктивності. Натомість, поведінка методом спроб і помилок та рівень фрустрації залежали від базових знань, а не від віку. Також було виявлено, що у людей похилого віку більше проблем із зручністю використання мобільних телефонів (Ziefle & Bay, 2005). Досліджуючи використання мобільних телефонів людьми похилого віку та характеристики зручного для людей похилого віку мобільного телефону, Курняван (2008) виявив, що люди похилого віку відчувають страх перед наслідками використання незнайомих технологій.

Важливість врахування, наприклад, статі була показана [13]. Ці дослідження показують, наприклад, що чоловіки, як правило, є більш абстрактними учнями, більш інтуїтивними та неорієнтованими, тоді як жінки більше стурбовані результатами, більш аналітичні та організовані. Крім того, роблять висновок, що чоловіки поводилися активніше, ніж жінки, у формальному середовищі онлайн-навчання, наприклад, надсилали більше та довших повідомлень. У нещодавньому дослідженні [10] було виявлено, що жінки демонстрували вищу комп'ютерну тривожність, ніж чоловіки, що означає, що чоловіки більш впевнені в собі, а жінки більш вагаються. Гендерні відмінності також виявлені в бажаних елементах дизайну, наприклад, жінки зосереджуються на тактильних засобах, а чоловіки - на перцептивних [12].

Міжкультурний дизайн інтерфейсів зараз поширений у промисловості та виробництві. Така тенденція спонукає до досліджень щодо обґрунтованості міжкультурного дизайну, впливу культурних відмінностей на поведінку користувачів під час взаємодії, а також того, як враховувати та враховувати культурні відмінності в дизайні інтерфейсів тощо. Завжди вважається, що люди мають подібні основні психологічні характеристики, тобто люди в усьому світі сприймають та міркують однаково [15] зазначили, що культурні відмінності впливають на застосування людьми бажаних навичок та стратегій до когнітивних процесів у кожній конкретній ситуації, навіть якщо люди мають подібні основні когнітивні функції. Як показують переконливі докази, те, як люди з різних демографічних регіонів світу сприймають об'єкти та ситуації,

формується культурно-історичними відмінностями у фізичному середовищі, вихованні, освіті та соціальній структурі [15]/

Пов'язуючи методологію оцінки зручності використання, культурні питання завжди виключаються з групи впливових факторів і розглядаються як такі, що не впливають на результати оцінки зручності використання в більшості досліджень [13], Протягом багатьох років проводилися дослідження для вивчення впливу впливових факторів на результати оцінки зручності використання, таких як вибір сценаріїв завдань, кількість учасників тестування, вибір методологій, вибір місць тестування тощо. Однак менше уваги приділяється культурним впливам на процес оцінювання, наприклад, чи впливає культурний фон учасників тестування на результати оцінювання, і чи повинен вибір сценаріїв завдань та евристики інтерфейсу ігнорувати культурні впливи. Щодо розбіжностей між дослідженнями зручності використання, обговорення рідко ведуться з точки зору культурних впливів. [18] провели глибокий аналіз культурного пізнання в оцінці зручності використання та проілюстрували вплив культурного пізнання на чотири центральні елементи методу мислення вголос (ТА): (1) інструкції та завдання; (2) вербалізації користувача; (3) сприйняття користувача оцінювачем та (4) загальні стосунки між користувачем та оцінювачем. На завершення, наголошується на деяких важливих висновках, таких як важливість відповідності презентації завдання культурному походженню користувачів, різний вплив мислення вголос на виконання завдання між жителями Сходу та Заходу, відмінності в невербальній поведінці, що впливають на виявлення проблем зручності використання, та складність загальних стосунків між користувачем та оцінювачем з різним культурним походженням.

У літературі «експертиза» визначається як механізм, що лежить в основі вищих досягнень експерта, тобто «той, хто набув спеціальних навичок або знань з певного предмета завдяки професійній підготовці та практичному досвіду» [17], або «експертні знання чи вміння, особливо в певній галузі» (Оксфордський словник для учнів підвищеного рівня, 1995). Однак під «експертизою

користувачів» ми завжди маємо на увазі спеціальні навички чи знання користувачів, які набуваються шляхом навчання, вивчення або практики. Очевидно, що експертиза користувачів пов'язана з їхньою практикою або досвідом роботи з певною системою чи предметом і може розглядатися як наслідок здатності користувачів до широкої адаптації до фізичного та соціального середовища. Досвід користувачів із певною системою чи предметом – це вимір, який зазвичай згадується під час обговорення експертних знань користувачів [6] Фактично, відмінності у досвіді користувачів є практичним питанням у взаємодії людини з машиною (HMI) або взаємодії людини з комп'ютером (HCI). Розвиток експертних знань користувачів часто відбувається завдяки тривалим періодам цілеспрямованої практики. Певною мірою досвід користувачів із певною системою чи предметом може значно сприяти їхньому набуттю знань. У багатьох дослідженнях користувачів класифікують на різні категорії відповідно до їхнього різного досвіду. Наприклад, [5] виділив три поширені класи користувачів за шкалою досвіду користувачів:

(1) користувачі-початківці — «користувачі, які знають завдання, але мало або взагалі не знають системи»;

(2) обізнані періодичні користувачі — «користувачі, які знають завдання, але через нечасте використання можуть мати труднощі зі запам'ятовуванням синтаксичних знань про те, як виконувати свої цілі»;

(3) досвідчені часті користувачі — «користувачі, які мають глибокі знання про завдання та пов'язані з ними цілі, а також дії, необхідні для досягнення цих цілей».

Класифікація [8] подібна до класифікації [4], тобто користувачів можна класифікувати як новачків або експертів, або як щось посередині. Порівняння між новачками та досвідченими користувачами ніколи не можна ігнорувати під час вивчення відмінностей у досвіді користувачів. Хоча деякі дослідники, такі як [4,8] надають теоретичні визначення категорій досвіду користувачів (тобто новачок, періодичний користувач та досвідчений користувач), межі між категоріями користувачів насправді дуже розмиті. Одна з причин полягає в тому,

що новачки перетворяться на досвідчених користувачів після певного навчання та практики. Розвиток досвіду — це процес навчання з поступовим набуттям навичок між новачками та експертами. Це означає, що категорії досвіду користувачів є динамічними та заснованими на статусі. Набуття навичок користувачами зазвичай пропорційне часу та практиці. Однак майже неможливо розрізнити категорії користувачів за допомогою точної дати/часу або точного обсягу практики. Категоризація може ґрунтуватися лише на поточному рівні навичок користувачів. Поширеною ідеєю є те, що експерти, здається, мають менше помилок і менший рівень їх серйозності, ніж новачки [16]. Крім того, ментальні моделі експертів є більш абстрактними, а моделі новачків здаються більш конкретними щодо рівнів знань [16]. Стверджується, що досвідчені користувачі краще виявляють недоліки дизайну [16] розуміють функціональні зв'язки [12] та визначають стратегії вирішення проблем [1]. Прості відмінності між новачками та експертами також можна знайти в деяких об'єктивних показниках продуктивності, таких як рівень помилок під час виконання завдання, витрати часу на виконання завдання, вирішення проблем, прийняття рішень та судження. Такі відмінності можна побачити в багатьох галузях. Наприклад, досвідчені програмісти завжди можуть писати комп'ютерні програми більш лаконічно та логічно, ніж програмісти-початківці; Лікарі-експерти майже завжди краще вміють правильно діагностувати захворювання, ніж лікарі-початківці; пілоти-експерти, як правило, вміліші в точному оцінюванні ситуації та реагуванні на випадкові події, ніж пілоти-початківці.

Досвід користувача проти ментальних моделей

У HCI ментальні моделі часто згадуються в дизайні інтерфейсу та тісно пов'язані зі зручністю використання; це пов'язано з їхньою головною роллю в когнітивних процесах та прийнятті рішень під час процесу взаємодії. За допомогою ментальних моделей дизайнери можуть отримати чітке уявлення про те, чого намагаються досягти користувачі, а потім відповідно та ефективно узгодити стратегію дизайну з поведінкою користувачів. Ментальні моделі завжди позначаються як психологічні або внутрішні репрезентації реальності,

які люди використовують для розуміння конкретних явищ. Поняття «ментальної моделі» вперше було сформульовано [2] з припущенням, що люди покладаються на ментальні моделі у своїй діяльності. Ментальна модель відображає справжнє коріння людської поведінки, філософії та емоцій щодо того, як виконувати завдання. У світлі визначення [3], ментальні моделі розкривають процес, у якому люди обробляють та вирішують завдання дедуктивного мислення, і вони не обов'язково можуть бути більш складними або відображати реальні випадки. [9] зазначив, що багато існуючих теорій ментальних моделей підтверджуються такими припущеннями: (1) користувачі формують ментальну модель внутрішньої роботи комп'ютерних систем, з якими вони взаємодіють; (2) зміст та структура цієї ментальної моделі впливають на те, як користувачі взаємодіють із системою; (3) вибір того, яка інформація про систему подається користувачам, і як вона подається, може впливати на зміст та структуру ментальної моделі; (4) більш детальні знання про те, як користувачі конструюють, викликають та адаптують ментальні моделі, можуть бути використані для надання рекомендацій щодо інтерфейсів користувача та навчання користувачів, що може допомогти користувачам формувати відповідні моделі. Загальна точка зору, виходячи з існуючих теорій ментальних моделей, полягає в тому, що люди можуть створювати ментальні моделі шляхом сприйняття, уяви або розмови. Багато дослідників присвячені дослідженню того, як конструюються ментальні моделі. Однак немає згоди щодо того, як саме конструюються ментальні моделі. Між різними теоріями існують деякі розбіжності, такі як структура ментальних моделей, як ментальні моделі впливають на взаємодію між користувачем і системою, і як конструюються ментальні моделі [3] зазначив, що ментальні моделі традиційно характеризуються двома способами, тобто як побудова ментальної моделі в конкретній ситуації та як активація з узагальнених знань. Ментальні моделі можуть бути неповними, простими, абстрактними та навіть загальними. Вони також можуть постійно розвиватися. Чим детальніші та правильніші ментальні моделі, тим більшого успіху досягнуть користувачі під час взаємодії з машинами чи пристроями.

Питання ментальних моделей є важливими та корисними для розуміння того, як люди обробляють інформацію в різних середовищах. Довгий час вважалося, що користувачі використовують певний тип моделі під час взаємодії з машинами. [22] зазначив, що ментальні моделі забезпечують «прогнозну та пояснювальну силу» для розуміння взаємодії. [22] визначив ментальну модель дизайнера як концептуальну модель. Кілька років по тому [22] запропонував діаграму та розрізнив моделі дизайнерів (тобто модель дизайну) та ментальні моделі користувачів (тобто модель користувача). Моделі дизайнерів завжди вважаються точним, послідовним та повним представленням цільової системи, тоді як моделі користувачів можуть бути обмеженими та частковими ментальними моделями моделей дизайнерів. За умови, що ментальна модель користувача відповідає моделі дизайнера, можна досягти зручності використання та зменшити кількість помилок. Було проведено багато досліджень, щоб з'ясувати, чи дійсно користувачі мають та використовують ментальні моделі під час взаємодії з пристроями та системами [21]. Спостерігаючи за роботою користувачів у системі та порівнюючи відмінності у здібностях до вирішення проблем між новачками та експертами в певній області, більшість досліджень роблять висновок про існування ментальних моделей. Ментальна модель користувачів впливатиме на успішність їхньої роботи. [21] стверджував, що неточні ментальні моделі складніших систем (наприклад, літаків, ядерних реакторів) можуть призвести до катастрофічних аварій. [21] зазначив, що помилки в рішеннях не є фактичними помилками, а нормальними наслідками ментальної моделі оператора.

Звертаючись до проблеми експертної компетентності користувача, актуальним питанням є те, чи мають новачки та досвідчені користувачі однакові чи різні ментальні моделі під час взаємодії, і як їхні ментальні моделі сприяють їхній роботі. [27] досліджував, як новачки та експерти міркують про фізичну ситуацію, і виявив, що модель новачка представляє об'єкти у світі та симулює процеси, що відбуваються в реальному часі, тоді як модель експерта представляє дуже абстрактні відносини та властивості. Загальною ідеєю багатьох дослідників

є те, що ментальні моделі експертів є більш абстрактними, а моделі новачків здаються більш конкретними щодо рівнів знань [25] використовував термін «макромодель» для опису багатших та абстрактніших моделей експертів. Експерти, здається, мають менше помилок та меншу серйозність помилок, ніж новачки, у роботі [27]. Щодо здібностей до вирішення проблем, новачки були паралізовані своєю здатністю вирішувати проблеми, тоді як експерти мали кілька стратегій для вирішення проблем [22] Наприклад, у дослідженні комп'ютерного програмування [23] виявив, що досвідчені розв'язувачі проблем витягували додаткові знання зі своїх складніших ментальних моделей для вирішення завдань, тоді як новачки зосереджувалися на поверхневих особливостях проблеми. Новачки міркують на основі ментальних симуляцій, які вимагають побудови моделей, що представляють типові послідовності дій [20] Однак, залишається багато пов'язаних питань, які залишаються незрозумілими або ще не вирішеними - наприклад, під час взаємодії з невідомою або новою системою/пристроєм, скільки допомоги досвідчені користувачі можуть отримати від своїх існуючих ментальних моделей для побудови нових ментальних моделей, і чи мають досвідчені та новачки різні ментальні моделі, коли вони орієнтуються в невідомій системі/пристрої, нехтуючи складними рівнями інтерфейсів. Крім того, дослідників дуже хвилюють питання, чи може стара ментальна модель досвідчених користувачів адаптуватися до нового контексту взаємодії, як така адаптація сприяє або впливає на їхню ефективність взаємодії в новому контексті, а також які зовнішні фактори можуть порушити адаптацію.

Різні типи або марки машин/систем можуть бути як схожими, так і відрізнятися одна від одної в деяких аспектах інтерфейсу, наприклад, з точки зору значків, символів, розташування інформаційного дисплея, меню та термінології. Подібність між подібними машинами/системами може призвести до плутанини для користувачів під час взаємодії через відсутність чітких меж між ментальними [25] Хоча основна теорія роботи з машинами більш-менш однакова, існують дві типові теоретичні погляди, що ілюструють, як ментальні

моделі досвідчених користувачів сприяють їхній навігації під час взаємодії. Різниця між цими двома теоретичними поглядами полягає в тому, чи залежать досвідчені користувачі від організації інтерфейсу під час їхньої взаємодії. Наприклад, [22] запропонували Теорію когнітивної гнучкості (ТКГ), щоб підкреслити адаптацію ментальних моделей досвідчених користувачів до нового та невідомого інтерфейсу, маючи на увазі, що досвідчені користувачі перевершують новачків завдяки адаптації їхньої ментальної моделі до змін нового або невідомого інтерфейсу/системи. Однак, теорія гіпотези налаштування обмежень (CAH) вказує на те, що краща продуктивність досвідчених користувачів пов'язана з організацією інтерфейсу, що означає, що досвідчені користувачі перевершать новачків, якщо новий та невідомий інтерфейс буде схожим на старі інтерфейси. Насправді, така схожість та варіації в організації інтерфейсів є непередбачуваними та контролюються виробниками. На основі систематичного дослідження досвіду користувачів у різних ситуаціях взаємодії було виявлено, що досвідчені користувачі, здається, покладаються на свою останню ментальну модель у вирішенні проблем, коли новачки та досвідчені користувачі взаємодіють з новим та невідомим типом або маркою простої машини/системи. Це означає, що досвідчені користувачі схильні покладатися на свої старі ментальні моделі для дослідження нового та невідомого інтерфейсу. Зіткнувшись з новою термінологією у взаємодії, зусилля досвідчених користувачів щодо вирішення проблеми, здається, ґрунтувалися на семантичній схожості, тоді як зусилля новачків здавалися безцільними [19]. З результатів цього поточного дослідження можна зробити висновок, що навігація досвідчених користувачів базується на їхніх старих ментальних моделях під час взаємодії з новим та невідомим інтерфейсом. Початківці мають можливість досліджувати або орієнтуватися в інтерфейсі без будь-яких існуючих старих ментальних моделей. [19] вказав на дві сторони старих ментальних моделей досвідчених користувачів, тобто існуючі старі ментальні моделі можуть як користуватися, так і перешкоджати взаємодії досвідчених користувачів з новим або невідомим інтерфейсом. Позитивна та негативна роль старих ментальних

моделей експертів залежить від ступеня подібності або відмінності між старим та новим/невідомим інтерфейсами.

З одного боку, досвідчені користувачі можуть отримати вигоду у своїй взаємодії, коли їхні старі ментальні моделі відповідають дизайну нового інтерфейсу. З іншого боку, коли старі ментальні моделі досвідчених користувачів не відповідають новому дизайну, досвідчені користувачі можуть мати проблеми у своїй взаємодії або зіткнутися з невдачами в дослідженні, якщо вони занадто твердо дотримуються своїх старих ментальних моделей. Щоб уникнути цього негативного ефекту, важливо, щоб розробники та виробники враховували питання стандартизації в дизайні. [19] зазначили, що термінологія є серйозною проблемою при використанні одних і тих самих медичних пристроїв від різних виробників. Такі термінологічні проблеми можуть призвести до плутанини у досвідчених користувачів у їхніх здогадках та дослідженні, що впливає на ефективність взаємодії досвідчених користувачів. У реальних медичних контекстах часто можна побачити, що одні й ті ж пристрої, розроблені та виготовлені різними виробниками, використовуються в одних і тих самих відділеннях лікарень. Якщо існує велика різниця в термінології між однаковими пристроями різних марок або типів, то термінологічна проблема принесе непотрібне роздратування та стрес медичному персоналу, і це навіть може призвести до потенційних ризиків у їхній повсякденній роботі.

1.2 Модель експертності користувачів проти складності інтерфейсів користувача

Вплив різниці в досвіді користувачів на продуктивність взаємодії слід вивчати разом із врахуванням складності інтерфейсів. Досі не існувало чітких або загальноприйнятих рекомендацій чи принципів щодо класифікації або визначення складності інтерфейсів користувача. Можуть існувати різні способи класифікації рівнів складності різних інтерфейсів. З практичної точки зору, Лю (2009) запропонував спосіб визначення складних рівнів інтерфейсів користувача

на основі п'яти критеріїв: (1) легкість маніпулювання, яку можна побачити в процесі навчання та тренінгів; (2) ієрархія завдань як за широтою, так і за глибиною; (3) кількість інформації в меню або спливаючих вікнах; (4) кількість елементів у меню та спливаючих вікнах; та (5) кількість когнітивних та фізичних ресурсів, необхідних для роботи. Такий спосіб категоризації головним чином базується на кількості функцій та підфункцій , ієрархічних рівнях систем меню та часі навчання. Обґрунтування категоризації було успішно використано для класифікації медичних інтерфейсів користувача в медичних галузях, які вимагають вищої та суворішої безпеки на роботі. Це можна побачити в дослідженнях взаємозв'язку між відмінностями в досвіді користувачів та складністю медичних інтерфейсів користувача.

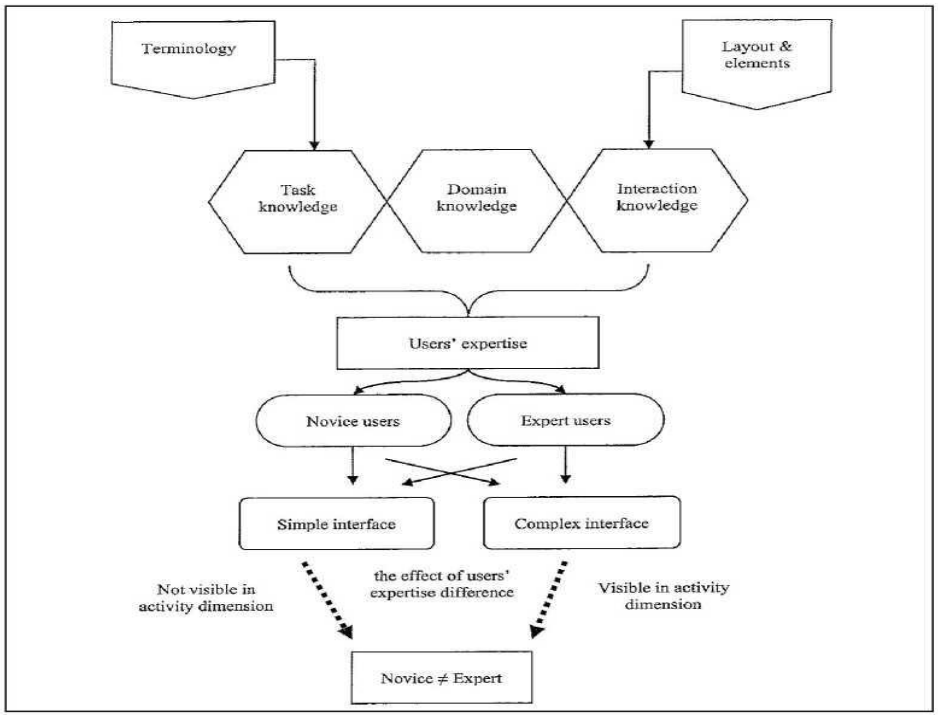

Рисунок 1.1 - Модель експертності користувачів у порівнянні зі складністю медичних інтерфейсів користувача

Щодо взаємодії з медичними пристроями, [19] запропонував модель (рис. 1), яка ілюструє досвід користувачів у порівнянні зі складністю медичних інтерфейсів користувача. У моделі за основу для аналізу та класифікації досвіду

користувачів було взято «Куб користувача» [25]. Три виміри, за якими класифікується досвід користувача, інтерпретуються як: (1) знання взаємодії, тобто загальні знання про взаємодію з певним машинним інтерфейсом та відповідними елементами (наприклад, розпізнавання функції системи меню, макета та елементів інтерфейсу, кнопок, значків тощо); (2) знання завдання, тобто знання області завдання, що вирішується певним інтерфейсом/системою (наприклад, термінологія); та (3) знання предметної області, тобто теоретичні знання або базова теорія про виконання завдання, яка не залежить від конкретного продукту або системи (наприклад, знання про те, як діяти, щоб впоратися з певним станом або завданням). Для взаємодії користувача з (медичним) пристроєм у медичній галузі три виміри, за якими класифікується досвід користувача, інтерпретуються відповідно як: знання взаємодії щодо конкретного медичного пристрою, знання завдання, що вирішується певним медичним пристроєм, та знання предметної області щодо конкретного медичного лікування. Таким чином, досвідчені користувачі визначаються як користувачі, які мають багаті знання взаємодії, знання завдань та знання предметної області конкретної системи, і вміють отримувати та використовувати ці знання для досягнення цілей або завдань у взаємодії. Аналогічно, новачки визначаються як користувачі, які ще не знайомі з певною системою, мають менший досвід використання/взаємодії та виглядають невмілими у використанні та отриманні знань для досягнення цілей та завдань у взаємодії.

Одна з проблем, ілюстрована цією моделлю, полягає в взаємозв'язку між інтерфейсом та різними компонентами або вимірами досвіду користувачів. Очевидним атрибутом інтерфейсів, що впливає на знання користувачами завдань, є термінологія, що використовується (тобто назви функцій, концепцій або значків). Однак очевидними атрибутами інтерфейсів, що впливають на знання взаємодії користувачів, здається, є макет та елементи (наприклад, структура меню та способи взаємодії). По-друге, модель показує, що вплив досвіду користувачів на поведінку користувачів відрізняється залежно від рівня складності медичних інтерфейсів користувача. Така модель надає інформацію

про відмінності між початківцями та досвідченими користувачами у вимірі активності. Під час взаємодії з простим медичним інтерфейсом користувача досвідчені користувачі не перевершують початківців за продуктивністю. Іншими словами, досвідчені користувачі схожі на початківців з точки зору виконання завдань. Однак досвідчені користувачі перевершили початківців під час взаємодії зі складним медичним інтерфейсом користувача, що означає, що вплив різниці в досвіді користувачів видно у вимірі активності. Складність складних інтерфейсів користувача полегшує початківцям загубитися в навігації. [19] зазначив, що користувачі-початківці, здається, вибирають кнопки або дії більш-менш випадково під час навігації, тоді як досвідчені користувачі базують свою взаємодію на міркуваннях та прийнятті рішень, що відображають їхні професійні навички. Хоча ця модель була розроблена на основі досліджень зручності використання в медичній галузі, результати та висновки можна використовувати або застосовувати й в інших галузях, таких як переробна промисловість, яка вимагає високого рівня безпеки та високого рівня знань/навичок користувачів.

1.3 Вплив досвіду користувача на оцінку зручності використання

Оцінка зручності використання є важливим кроком у розробці та вдосконаленні дизайну для UCD та інтерактивного дизайну. Для дизайну інтерфейсу оцінка зручності використання необхідна на різних етапах усього процесу проектування, щоб уникнути проблем із зручністю використання з самого початку. Тестування зручності використання [27] є поширеним методом оцінки зручності використання продукту або системи. У низці попередніх досліджень досліджувався вплив різних методологічних підходів на результат тесту зручності використання. Наприклад, порівнювалися різні методи збору даних, включаючи порівняння між теоретичними та емпіричними оцінками. Тести зручності використання в лабораторних умовах порівнювалися з тестами в реальних умовах [21] Був досліджений вплив кількості учасників тестів зручності використання [21], а також досліджено результати окремих

користувачів проти тих, хто співпрацює, як учасників тестування [13]. Також був оцінений вибір завдань оцінювання. Крім того, було розглянуто рівень втручання експериментатора та аспект оцінювачf. Як дедалі важливіший аспект досліджень зручності використання, профіль користувача привертає увагу. Отримуючи знання про вік та стать цільових користувачів, а також, наприклад, про їхню освіту та культурне походження, вважається можливим краще передбачити потенційні труднощі, з якими користувачі можуть зіткнутися під час вивчення та взаємодії з інтерфейсом.

Комп'ютерні технічні пристрої або машини різних брендів або від різних виробників надають споживачам або покупцям можливість вибирати варіанти, які є найбільш підходящими з точки зору прибутку від бізнесу або практичних міркувань у сучасному маркетингу. Наприклад, лікарням доводиться час від часу оновлювати або купувати нове обладнання, щоб відповідати вимогам лікування, а іноді їм доводиться розглядати можливість оновлення та купівлі обладнання різних брендів/типів або, можливо, від різних виробників. Практичне питання полягає в тому, чи можуть досвідчені користувачі передавати свій досвід, щоб полегшити свою взаємодію з новим брендом/типом комп'ютерних систем, тобто, якщо досвідчених користувачів просять виконувати свої звичні або рутинні завдання, чи можуть вони адаптувати свою продуктивність або можливості до нових брендів/типів комп'ютерних систем. Крім того, як дизайнери повинні залучати користувачів з різним рівнем досвіду до процесу проектування, а також вибирати учасників тестування на етапі оцінки зручності використання, щоб оптимально покращити процес проектування? Отже, в цьому розділі ми зосереджуємося на обговоренні досвіду користувачів та його впливу на оцінку зручності використання.

У традиційному обґрунтуванні тестування зручності використання, новачкам та експертам завжди дають окремі тести та різні завдання; наприклад, тести за участю новачків зосереджені на навчанні, тоді як тести з досвідченими користувачами найчастіше зосереджені на оптимальному використанні [10]. Різниця в фокусі тестування неминуче передбачає порівняння на різних рівнях

користувачів [10]. Дослідження, проведене дослідниками з Гонконгського університету науки і технологій, показало, що тестування та аналіз продуктивності новачків та досвідчених користувачів в одному тесті забезпечує додатковий рівень інформації, який не забезпечує тестування двох окремих елементів [9]. На основі вивчення рекомендацій щодо тестування зручності використання та рекомендацій з популярних довідників з зручності використання, таких як [12], підтверджується поширена точка зору, тобто тестування новачків та експертів з різних причин та використання різних підходів до збору інформації. Щоб отримати уявлення про повний спектр зручності використання, важливо отримати дані від користувачів усіх рівнів. Хоча низка досліджень стверджує про важливість врахування досвіду користувачів у тестах зручності використання [9], мало досліджень систематично досліджували вплив різниці в досвіді користувачів на результати тестування, коли розглядалися різні категорії користувачів. Можна очікувати, що рівень досвіду користувачів впливатиме на проблеми, з якими вони можуть зіткнутися під час навчання взаємодії або взаємодії з інтерфейсом користувача, тому можна припустити, що цей самий аспект впливає і на результати тесту зручності використання.

Найновішим дослідженням проблем користувацького досвіду є емпіричне дослідження впливу користувацького досвіду та точності прототипу на результати тесту зручності використання, У цьому дослідженні користувацький досвід (експерт проти новачка) та точність прототипу (паперовий прототип, 3D-макет та повністю функціональний пристрій) маніпулювалися як незалежні змінні в міжсуб'єктному дизайні 2 x 3. Результати показали, що експерти мали більше проблем із зручністю використання, але з меншим рівнем серйозності, ніж новачки. Прототипи зі зниженою точністю виявилися в основному придатними для прогнозування зручності використання реальних продуктів, що дає наслідки для проблем із точністю прототипу під час проведення тестів зручності використання. [19] проаналізував та узагальнив вплив досвіду користувачів на результати тестів зручності використання під час

взаємодії з користувацькими інтерфейсами різного рівня складності (тобто простий інтерфейс та складний інтерфейс). Внесок користувачів з різним досвідом (тобто новачків та досвідчених користувачів) під час тестів зручності використання був проаналізований та порівняний за двома вимірами, тобто виміром активності, що стосується активності користувачів під час виконання завдання (наприклад, дії, допущені помилки), та виміром усного пояснення, що стосується усного пояснення користувачів або суб'єктивних думок (наприклад, їхньої презентації пропозицій щодо редизайну). Інформація, отримана з результатів тестування, яка виявила внесок користувачів з різним досвідом, була класифікована відповідно до цих двох вимірів. При аналізі активності користувачів враховувалися як кількісні дані (наприклад, кількість помилок та час виконання завдання), так і якісні дані (наприклад, причини помилок на основі аналізу усних протоколів), тоді як при аналізі усних пояснень користувачів враховувалися лише якісні дані (наприклад, способи подання інформації, джерела прийняття рішень щодо пропозицій щодо редизайну, зміст/обсяг пропозицій).

Було виявлено, що кількісний аналіз, наприклад, часу виконання завдання та/або кількості помилок, лише на досить поверхневому рівні відображає відмінності в досвіді користувачів. Це означає, наприклад, що між новачками та досвідченими користувачами може не бути відмінностей під час взаємодії з простим інтерфейсом, але відмінності будуть очевидними при взаємодії зі складнішим інтерфейсом. Однак результати, що стосуються якісного аналізу словесних пояснень та причин помилок, натомість підкреслили відмінності між новачками та досвідченими користувачами, характеризуючи основу для рішень, що лежать в основі певних дій, а також представлення пропозицій щодо редизайну. Тому кількісного аналізу недостатньо для дослідження та розуміння відмінностей у досвіді користувачів та їх впливу на результат тесту зручності використання. Недостатність знань предметної області та взаємодії, наприклад, постійно визначається як типова причина помилок для користувачів-початківців. З іншого боку, відмінності в організації інформації між раніше знайомими

користувацькими інтерфейсами та інтерфейсами, з якими взаємодіяли в тестах зручності використання, впливали на виконання завдань досвідченими користувачами. Як наслідок, досвідчені користувачі допускали помилки, пов'язані із завданнями, через проблеми з термінологією та помилки, пов'язані з взаємодією, через свою «стару» ментальну модель взаємодії з користувацьким інтерфейсом. Деякі помилки через брак уваги були виявлені в обох групах користувачів. Однак, досвідчені користувачі допускали такі ненавмисні помилки через негативний вплив задоволеності досвідчених користувачів своїм рівнем знань. Початківці допускали кілька таких ненавмисних помилок через незвичайні зовнішні фактори (наприклад, тимчасову та непередбачувану втому під час тестів). Якісний аналіз усних висловлювань також виявив деякі типові відмінності між новачками та досвідченими користувачами у прийнятті рішень, презентації та судженнях, що означало, що досвід використання досвідченими користувачами та досвід взаємодії з новачками різниться у внеску в дизайн та розробку продукту.

1.4 Висновки по розділу

Еволюція підходів до проектування інтерфейсів засвідчує перехід від технікоцентричного до користувача орієнтованого підходу, де ключову роль відіграє глибоке розуміння потреб, особливостей та досвіду користувачів. Врахування індивідуальних відмінностей та активне залучення користувачів до процесу проектування стало основою ефективного. Модель Лю демонструє, що відмінності у досвіді користувачів виявляються лише при взаємодії зі складними інтерфейсами. Досвід користувача суттєво впливає на результати тестування зручності використання: тоді як кількісні метрики часто не виявляють різниці, якісний аналіз показує, що новачки та експерти по-різному взаємодіють із системами та вносять різний внесок у процес редизайну. Це підкреслює необхідність залучення користувачів із різним рівнем досвіду для повноцінної оцінки.

РОЗДІЛ 2. ОГЛЯД ІНСТРУМЕНТІВ ПРОЄКТУВАННЯ ІНТЕРФЕЙСУ КОРИСТУВАЧА

2.1 Важливість інструментів інтерфейсу користувача

Використання програмних інструментів для роботи з інтерфейсом користувача має багато переваг. Їх можна розділити на дві основні групи. По-перше, якість отриманих інтерфейсів користувача може бути вищою. Це пояснюється наступним:

• Прототипи та впровадження дизайну можливі швидко, можливо, навіть до написання коду програми. Це, у свою чергу, дозволяє швидше прототипування, а отже, і більше ітерацій ітеративного дизайну, що є вирішальним компонентом досягнення високоякісних користувацьких інтерфейсів.

• Надійність користувацького інтерфейсу буде вищою, оскільки код для користувацького інтерфейсу створюється автоматично зі специфікації вищого рівня.

• Різні програми, ймовірніше, матимуть узгоджені інтерфейси користувача, якщо вони створені з використанням той самий інструмент інтерфейсу користувача.

• Різним спеціалістам буде легше залучати до розробки користувацького інтерфейсу різноманітні фахівці, ніж повністю створювати користувацький інтерфейс програмістами. Можуть бути залучені графічні дизайнери, когнітивні психологи та фахівці з юзабіліті. Зокрема, професійні дизайнери користувацьких інтерфейсів, які можуть не бути програмістами, можуть відповідати за загальний дизайн.

• На інструмент можна витратити більше зусиль, ніж це практично можливо для будь-якого окремого користувацького інтерфейсу, оскільки інструмент використовуватиметься з багатьма різними програмами.

• Функції «Скасувати», «Довідка» та інші, ймовірно, будуть доступні, оскільки інструменти можуть їх підтримувати.

По-друге, код інтерфейсу користувача може бути простішим та економічнішим у створенні та підтримці. Це через наступне:

• Специфікації інтерфейсу можна легше представляти, перевіряти та оцінювати.

• Коду потрібно буде писати менше, оскільки більшість надається інструментами.

• Завдяки відокремленню компонента інтерфейсу користувача від програми буде забезпечено кращу модуляризацію. Це має дозволити змінювати інтерфейс користувача, не впливаючи на програму, а також широкий спектр змін у програмі (наприклад, зміна внутрішніх алгоритмів) має бути можливим без впливу на інтерфейс користувача.

• Рівень знань програмування дизайнерів та реалізаторів інтерфейсів може бути нижчим, оскільки інструменти приховують значну частину складнощів базової системи.

• Перенести застосунок на різні апаратні та програмні середовища буде легше, оскільки залежності пристроїв ізольовані в інструменті користувацького інтерфейсу.

2.2 Огляд інструментів програмного забезпечення для інтерфейсу користувача

Оскільки програмне забезпечення для користувацького інтерфейсу так складно створити, не дивно, що люди вже давно працюють над створенням інструментів, які б у цьому полегшили роботу. Сьогодні багато з цих інструментів та ідей пройшли шлях від досліджень до комерційних систем, і їхня ефективність була повністю продемонстрована. Дослідницькі системи також продовжують швидко розвиватися, а моделі, які були популярними п'ять років тому, застаріли завдяки більш ефективним інструментам, змінам на комп'ютерному ринку та появі нових стилів користувацьких інтерфейсів, таких як портативні комп'ютери та мультимедіа.

Як показано на рисунку 1.1, програмне забезпечення інтерфейсу користувача можна розділити на різні рівні: віконну систему, інструментарій та інструменти вищого рівня. Звичайно, багато практичних систем охоплюють кілька рівнів. Система вікон підтримує розділення екрана на різні (зазвичай прямокутні) області, які називаються вікнами. Система X поділяє функціональність вікон на два шари: систему вікон, яка є функціональним або програмним інтерфейсом, та менеджер вікон, який є інтерфейсом користувача. Таким чином, система вікон надає процедури, які дозволяють програмі малювати зображення на екрані та отримувати вхідні дані від користувача, а менеджер вікон дозволяє кінцевому користувачеві переміщувати вікна та відповідає за відображення рядків заголовка, меж та значків навколо вікон. Однак багато людей та систем використовують назву «менеджер вікон» для позначення обох шарів, оскільки такі системи, як Macintosh та Microsoft Windows, не розділяють їх. У цій статті буде використано термінологію X та термін «система вікон», коли буде йшлося про обидва шари.

Зверніть увагу, що Microsoft плутано називає всю свою систему «Windows» (наприклад, «Windows 98» або «Windows XP»). Це включає багато різних функцій, які тут поділяються на частину операційної системи (яка підтримує керування пам'яттю, доступ до файлів, мережу тощо), віконну систему та інструменти вищого рівня. Поверх віконної системи знаходиться інструментарій, який містить багато часто використовуваних віджетів (також званих елементами керування), таких як меню, кнопки, смуги прокручування та поля введення тексту. Поверх інструментарію можуть бути інструменти вищого рівня, які допомагають дизайнеру використовувати віджети інструментарію. У наступних розділах кожен із цих компонентів обговорюється більш детально.

Віконні системи

Віконна система — це програмний пакет, який допомагає користувачеві контролювати та керувати різними контекстами, фізично розділяючи їх на різні частини одного або кількох екранів відображення. Хоча більшість сучасних систем надають інструментальні набори поверх віконних систем, як буде

пояснено нижче, інструментальні набори зазвичай стосуються лише малювання віджетів, таких як кнопки, меню та смуги прокручування. Таким чином, коли програміст хоче намалювати специфічні для програми частини інтерфейсу та дозволити користувачеві маніпулювати ними, інтерфейс віконної системи повинен використовуватися безпосередньо. Тому програмний інтерфейс віконної системи має значний вплив на більшість програмістів користувацького інтерфейсу. Перші віконні системи були реалізовані як частина однієї програми або системи. Наприклад, текстовий редактор EMAC, а також середовища програмування Smalltalk та DLISP мали власні віконні системи. Пізніші системи реалізували віконну систему як невід'ємну частину операційної системи, такі як Sapphire для PERQ, SunView для Suns, а також системи Macintosh та Microsoft Windows. Щоб дозволити різним віконним системам працювати в одній операційній системі, деякі віконні системи, такі як X та Sun's NeWS, працюють як окремий процес і використовують механізм міжпроцесного зв'язку операційної системи для підключення до прикладних програм.

Віконну систему можна логічно розділити на два шари, кожен з яких має дві частини (див. Рисунок 2.1). Віконна система, або базовий шар, реалізує основні функції віконної системи. Дві частини цього шару обробляють відображення графіки у вікнах (модель виводу) та доступ до різних пристроїв введення (модель введення), які зазвичай включають клавіатуру та вказівний пристрій, такий як миша. Основний інтерфейс базового шару є процедурним і називається інтерфейсом програмування додатків (API) віконної системи. Інший рівень віконної системи – це менеджер вікон або інтерфейс користувача. Він включає всі аспекти, видимі для користувача. Дві частини рівня інтерфейсу користувача – це презентація, яка складається з зображень, що відображається менеджером вікон, та команди, які відповідають за те, як користувач маніпулює вікнами та їхнім вмістом.

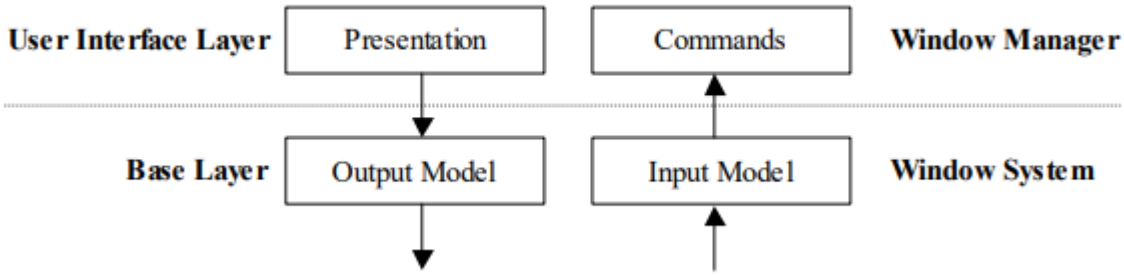

Рисунок 2.1 - Віконну систему можна розділити на два шари, які називаються базовим шаром або шаром віконної системи, та користувацький інтерфейс або рівень менеджера вікон.

Базовий рівень – це процедурний інтерфейс до віконної системи. У 1970-х та на початку 1980-х років існувала велика кількість різних віконних систем, кожна з яких мала різний процедурний інтерфейс (принаймні одна для кожної апаратної платформи). Люди, які писали програмне забезпечення, вважали це неприйнятним, оскільки вони хотіли мати можливість запускати своє програмне забезпечення на різних платформах, але їм довелося б переписувати значну кількість коду для конвертації з однієї віконної системи в іншу. Віконна система X була створена для вирішення цієї проблеми, надаючи апаратно-незалежний інтерфейс до віконної системи. X досягла успіху в цьому і витіснила всі інші віконні системи з ринку обладнання робочих станцій. X продовжує бути популярною як віконна система для Linux та всіх інших реалізацій Unix. На решті комп'ютерного ринку більшість машин використовують певну версію Microsoft Windows, а комп'ютери Apple Macintosh мають власну віконну систему.

Вихідна модель

Модель виводу – це набір процедур, які програма може використовувати для малювання зображень на екрані. Важливо, щоб весь вивід спрямовувався через віконну систему, щоб графічні примітиви можна було обрізати до меж вікна. Наприклад, якщо програма малює лінію, яка виходить за межі вікна, її потрібно обрізати, щоб вміст інших, незалежних вікон не перезаписувався. Більшість комп'ютерів оснащені графічним обладнанням, оптимізованим для

ефективної роботи з віконною системою. У ранніх віконних системах, таких як Smalltalk та Sapphire, основною операцією виведення була BitBlt (також звана «RasterOp», а тепер іноді «CopyArea» або «CopyRectangle»). Ці ранні системи в основному підтримували монохромні екрани (кожен піксель або чорний, або білий). BitBlt бере прямокутник пікселів з однієї частини екрана та копіює його в іншу частину. Для об'єднання значень пікселів прямокутників джерела та призначення можна вказати різні логічні операції. Наприклад, прямокутник джерела може просто замінити прямокутник призначення, або його можна об'єднати з місцем призначення за допомогою операції XOR. BitBlt можна використовувати для малювання суцільних прямокутників чорного або білого кольору, відображення тексту, прокручування вікон та виконання багатьох інших ефектів. Єдиною додатковою операцією малювання, яку зазвичай підтримували ці ранні системи, було малювання прямих ліній.

Пізніші віконні системи, такі як Macintosh та X, додали повний набір операцій малювання, таких як заповнені та незаповнені полігони, текст, лінії, дуги тощо. Їх неможливо реалізувати за допомогою оператора BitBlt. Зі зростанням популярності кольорових екранів та непрямокутних примітивів (таких як заокруглені прямокутники) використання BitBlt значно зменшилося. Зараз він в основному використовується для прокручування та копіювання зображень поза екраном на екран (наприклад, для реалізації подвійної буферизації). Деякі віконні системи дозволяли використовувати повну модель візуалізації Postscript для створення зображень на екрані. Postscript забезпечує незалежні від пристрою системи координат та довільні обертання та масштабування для всіх об'єктів, включаючи текст. Ще однією перевагою використання Postscript для екрана є те, що та сама мова може бути використана для друку вікон на папері (оскільки багато принтерів приймають Postscript). Sun створила версію, яка використовується у віконній системі NeWS, а потім Adobe (розробник Postscript) випустила офіційну версію під назвою «Display Postscript», яка використовувалася у віконній системі NeXT. Подібна модель візуалізації надається Java 2D, яка працює поверх (і приховує) вихідної моделі базової

віконної системи.

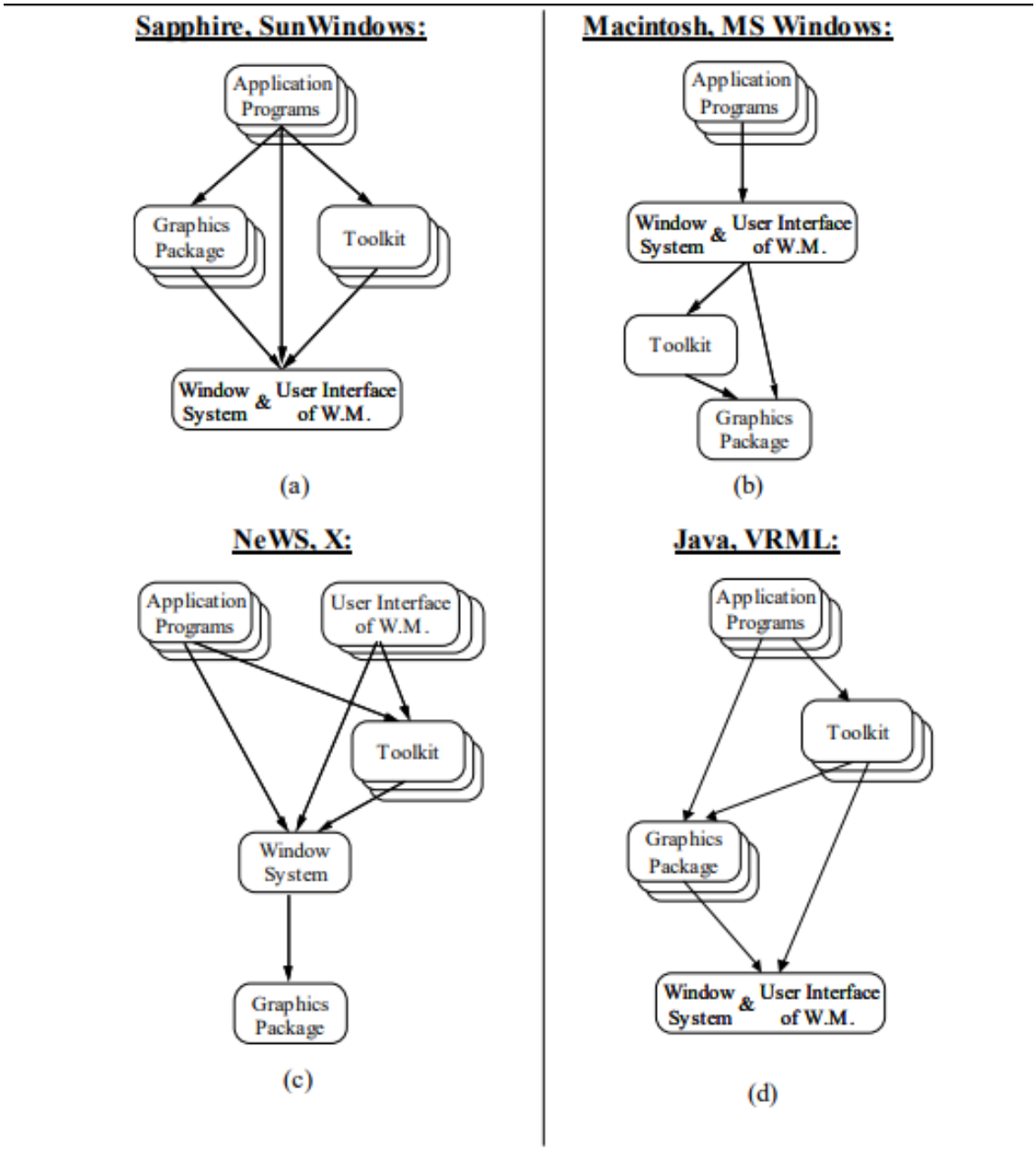

Рисунок 2.2 - Різні організації, які використовували віконні системи.

Коробки з додатковими межами представляють системи, які можуть бути замінені користувачами. Ранні системи (a) тісно пов'язували віконний менеджер і віконну систему і припускали, що складна графіка і набори інструментів будуть побудовані поверх. Наступним кроком у розробках (b) було включення графіки та інструментарію у віконну систему, щоб сам віконний менеджер мав більш досконалий вигляд, а програми були більш узгодженими. Інші системи (c) допускають різні віконні менеджери і різні набори інструментів, але при цьому

включають складні графічні пакети. Новіші системи (d) повертаються до початкового дизайну (a) і реалізують графіку та інструментарій поверх віконної системи.

Усі стандартні моделі виводу містять операції малювання лише для двовимірних об'єктів. Розширення для підтримки 3D-об'єктів включають PEX, OpenGL та Direct3D. PEX – це розширення віконної системи X, яке включає значну частину графічного стандарту PHIGS. OpenGL базується на програмному інтерфейсі GL, який багато років використовувався на машинах Silicon Graphics. OpenGL забезпечує певну машинну незалежність для 3D, оскільки він доступний для різних платформ X та Windows. Microsoft постачає власну модель 3D-графіки під назвою Direct3D як частину Windows. Як показано на рисунку 3, попередні віконні системи передбачали, що графічний пакет буде реалізовано за допомогою віконної системи (рисунок 2.2-a). Наприклад, графічний пакет CORE був реалізований поверх віконної системи SunView. Далі, такі системи, як Macintosh, X, NeWS, NeXT та Microsoft Windows, реалізували складну графічну систему як частину віконної системи (рисунок 2.2 b та c). Тепер, з Java2D та Java3D, а також веб-графічними системами, такими як VRML для 3D-програмування в Інтернеті, ми спостерігаємо повернення до моделі, подібної до рисунка 2.2-a, з графікою поверх віконної системи (див. рисунок 2.2-d).

Вхідна модель

Ранні графічні стандарти, такі як CORE та PHIGS, пропонували модель введення, яка не підтримувала сучасний стиль прямого маніпулювання інтерфейсами. У цих стандартах програміст викликає підпрограму для запиту значення віртуального пристрою , такого як локатор (положення вказівного пристрою), рядок (редагований текстовий рядок), вибір (вибір з меню) або вибір (вибір графічного об'єкта). Потім програма зупинялася, чекаючи на дію користувача. Це явно суперечить стилю без прямого маніпулювання , де користувач може вирішити, чи зробити вибір у меню, вибрати об'єкт або ввести щось. З появою сучасних віконних систем була запропонована нова модель: потік записів подій надсилається до вікна, яке наразі приймає вхідні дані.

Користувач може вибрати, яке вікно отримує події, використовуючи різні команди, описані нижче. Кожен запис події зазвичай містить тип і значення події (наприклад, яка клавіша була натиснута), вікно, на яке була спрямована подія, позначку часу та координати x та y миші. Віконна система ставить у чергу події клавіатури, події кнопок миші та події руху миші разом (разом з іншими спеціальними подіями), і програми повинні вилучати ці події з черги та обробляти їх. Дещо дивно, що, хоча в моделі виводу для віконних систем досягнуто значного прогресу (від BitBlt до складних 2D-примітивів та 3D), вхідні дані обробляються по суті так само, як і в оригінальних віконних системах, хоча з цією моделлю є деякі добре відомі невирішені проблеми:

- Немає можливості для спеціальних подій зупинки виводу (control-S) або переривання (control-C, command-dot), тому вони будуть поставлені в чергу разом з іншими подіями вводу.
- Той самий механізм подій використовується для передачі спеціальних повідомлень від віконної системи до програми. Коли вікно збільшується або стає відкритим, програму зазвичай потрібно повідомити, щоб вона могла налаштувати або перемалювати зображення у вікні. Більшість віконних систем повідомляють про це, додаючи спеціальні події до потоку подій, які потім має обробити програма.
- Програма завжди повинна бути готова приймати події для обробки переривань та запитів на перемальовування. Якщо ні, то довгі операції не можуть бути перервані, і на екрані можуть бути порожні області під час їх обробки.
- Модель залежить від пристрою, оскільки запис події має фіксовані поля для очікуваних вхідних подій. Якщо замість миші використовується тривимірний вказівний пристрій або пристрій з більшою, ніж стандартна кількість кнопок, то стандартний механізм обробки подій не зможе його обробити.
- Оскільки події обробляються асинхронно, існує багато умов змагання, які можуть призвести до розсинхронізації програм із віконною системою. Наприклад, у віконній системі X, якщо натиснути всередині вікна та

відпустити за його межами, за певних умов програма вважатиме, що кнопка миші все ще натиснута. Інший приклад полягає в тому, що запити на оновлення від віконної системи вказують прямокутник вікна, який потрібно перемалювати, але якщо програма змінює вміст вікна, на момент обробки події може бути перемалювана неправильна область. Ця проблема може виникнути під час прокручування вікна. Хоча ці проблеми відомі вже давно, досліджень нових моделей вхідних даних було проведено мало (винятком є модель Garnet Interactors).

У віконній системі X та NeWS весь зв'язок між програмами та віконною системою використовує міжпроцесний зв'язок через мережевий протокол. Це означає, що прикладна програма може знаходитися на іншому комп'ютері, ніж її вікна. У всіх інших віконних системах операції реалізуються шляхом безпосереднього виклику процедур менеджера вікон або через спеціальні пастки в операційній системі. Основною перевагою механізму X є те, що він спрощує використання кількох машин, при цьому всі їхні вікна відображаються на одній машині. Ще однією перевагою є те, що легше забезпечити інтерфейси для різних мов програмування: наприклад, інтерфейс C (який називається xlib) та інтерфейс Lisp (який називається CLX) надсилають відповідні повідомлення через мережевий протокол. Основним недоліком є ефективність, оскільки кожен запит вікна зазвичай кодується, передається на транспортний рівень, а потім декодується, навіть коли обчислення та вікна знаходяться на одній машині.

Інтерфейс користувача віконної системи дозволяє користувачеві керувати вікнами. У X користувач може легко перемикатися між інтерфейсами користувача, завершуючи один менеджер вікон та запускаючи інший. Деякі з оригінальних менеджерів вікон у X включали uwm (без рядків заголовка та рамок), twm, mwm (менеджер вікон Motif) та olwm (менеджер вікон OpenLook). Новіші варіанти включають повноцінні середовища робочого столу, які поєднують менеджер вікон з файловим браузером та іншими утилітами графічного інтерфейсу (щоб краще відповідати можливостям Windows та Macintosh). Два популярних середовища робочого столу - це KDE («K» Desktop

Environment» — http://www.kde.org/) з його менеджером вікон KWin та Gnome (http://www.gnome.org/) , яке надає різноманітні варіанти менеджера вікон. X надає стандартний протокол, через який програми та базовий рівень взаємодіють з менеджером вікон, так що всі програми продовжують працювати без змін, коли менеджер вікон перемикається. Наприклад, можна запускати програми, що використовують віджети Motif, усередині вікон, якими керує менеджер вікон KWin. Обговорення опцій інтерфейсів користувача віконних менеджерів було опубліковано раніше. Також відео «Усі віджети» містить 30-хвилинний сегмент, що демонструє багато різних форм інтерфейсів користувача віконних менеджерів. Деякі частини інтерфейсу користувача віконної системи, які іноді називають її зовнішнім виглядом та функціональністю, очевидно, можуть бути захищені авторським правом та запатентовані. Які саме частини – це дуже складне питання, і статус змінюється залежно від рішень у різних судових справах.

Зовнішній вигляд вікон визначає зовнішній вигляд екрана. Одним дуже важливим аспектом представлення вікон є те, чи можуть вони перекриватися. Перекриваючі вікна, які іноді називають перекритими вікнами, дозволяють одному вікну частково або повністю розташовуватися поверх іншого, як показано на малюнку. Рисунок 2.2. Це також іноді називають метафорою робочого столу, оскільки вікна можуть перекривати одне одного, як аркуші паперу можуть перекривати один одного на столі. Однак зазвичай метафора робочого столу має й інші аспекти, такі як представлення операцій з файлами таким чином, що імітує офісні операції, як це було виникло на офісній робочій станції Star. Інший варіант називається плитковими вікнами, що означає, що вікнам не дозволяється перекривати одне одного. Очевидно, що менеджер вікон, який підтримує перекриті вікна, також може дозволити їм розташовуватися поруч, але не навпаки. Тому менеджер вікон класифікується як «покритий», якщо він дозволяє вікнам перекриватися. Плитковий стиль був популярним деякий час і використовувався Ceda, ранніми версіями Star, Andrew і навіть Microsoft Windows. Дослідження навіть показало, що використання плиткових

вікон було ефективнішим для користувачів. Однак сьогодні плиткові вікна рідко можна побачити на звичайних віконних системах, оскільки користувачі зазвичай віддають перевагу перекриттям.

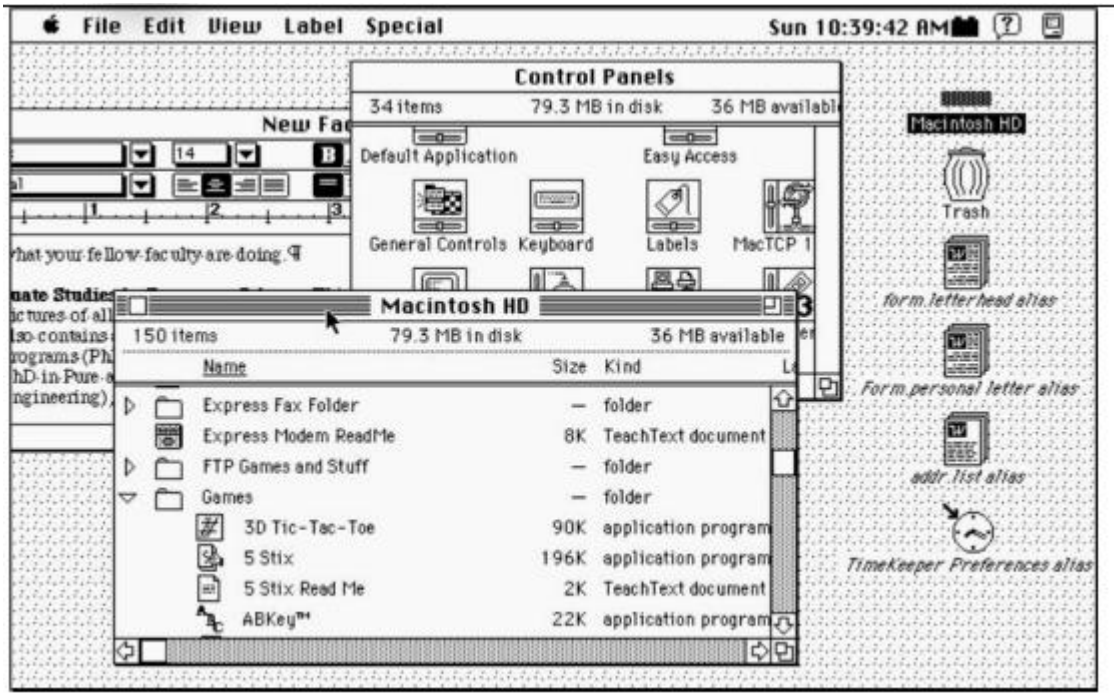

Рисунок 2.3 - Екран з оригінального Macintosh, на якому показано 3 вікна, що накривають одне одного, та деякі піктограми вздовж правого поля

Сучасні «браузери» для Всесвітньої павутини, такі як Netscape та Microsoft Internet Explorer, забезпечують віконне середовище всередині основної віконної системи комп'ютера. Новіші версії браузерів підтримують фрейми, що містять кілька прокручуваних панелей, які є формою плиткового вікна. Крім того, якщо завантажити програму, написану на Java (див. розділ про віртуальні інструменти нижче), вона може створювати кілька вікон, що перекриваються, як звичайні програми з графічним інтерфейсом. Ще одним важливим аспектом представлення вікон є використання піктограм. Це маленькі зображення, що представляють вікна (або іноді файли). Вони використовуються тому, що інакше вікон було б забагато для зручного розміщення на екрані та керування ними. Sapphire був першим менеджером вікон, який згрупував піктограми у вікно, що було підхоплено менеджером вікон Motif. Тепер «панель завдань» містить

піктограми та назви запущених і доступних процесів у Windows та інших сучасних менеджерах вікон. Інші аспекти представлення включають те, чи має вікно рядок заголовка чи ні, як виглядає фон (де немає вікон) та чи мають заголовок і межі області керування для виконання операцій з вікном.

Оскільки комп'ютери зазвичай мають кілька вікон і лише одну мишу та клавіатуру, користувач повинен мати спосіб контролювати, яке вікно отримує ввід з клавіатури. Це вікно називається фокусом введення (або клавіатури). Інший термін - слухач, оскільки він слухає введення користувача. Деякі системи називають фокус активним вікном або поточним вікном, але це невдалі терміни, оскільки в багатопроцесорній системі багато вікон можуть активно виводити інформацію одночасно. Менеджери вікон пропонують різні способи визначення та відображення того, яке вікно є слухачем. Найважливішими параметрами є:

- Введення тексту за допомогою миші, що означає, що користувач повинен клацнути кнопкою миші у вікні, перш ніж вводити в нього текст. Цей метод використовується в Macintosh та Microsoft Windows.
- Переміщення для друку, що означає, що для друку в вікні потрібно лише перемістити курсор миші. Зазвичай це швидше для користувача, але може призвести до того, що введення потрапить у неправильне вікно, якщо користувач випадково торкнеться миші.

Деякі віконні менеджери X (включаючи віконний менеджер Motif, mwm) дозволяють користувачеві вибирати бажаний метод. Однак цей вибір може суттєво вплинути на інтерфейс користувача програм. Наприклад, оскільки Macintosh вимагає введення тексту за допомогою клацання, він може забезпечити одне меню вгорі, і команди завжди можуть виконуватися у вікні, на якому знаходиться фокус. Завдяки переміщенню для введення тексту користувачеві, можливо, доведеться проходити через різні вікна (таким чином надаючи йому фокус) на шляху до верхньої частини екрана. Тому програми Motif повинні мати меню в кожному вікні, щоб команди знали, у якому вікні виконувати дії. Усі системи закритих вікон дозволяють користувачеві розмістити вікно вгорі (не закрите іншими вікнами), а деякі дозволяють розмістити вікно

внизу (закрите всіма іншими вікнами). Інші команди дозволяють змінювати розмір вікон, переміщувати їх, зменшувати до значка, робити їх повнорозмірними та знищувати.

2.3 Набори інструментів

Інструментарій – це бібліотека віджетів , які можуть викликатися прикладними програмами. Віджет (також званий елементом керування) – це спосіб використання фізичного пристрою введення для введення певного типу значення. Зазвичай віджети в інструментаріях включають меню, кнопки, смуги прокручування, поля введення тексту тощо. На рисунку 5 показано деякі приклади віджетів. Створення інтерфейсу за допомогою інструментарію можуть виконувати лише програмісти, оскільки інструментарії мають лише процедурний інтерфейс. Використання набору інструментів має перевагу в тому, що кінцевий інтерфейс користувача виглядатиме та працюватиме аналогічно іншим інтерфейсам, створеним за допомогою того ж набору інструментів, і кожній програмі не потрібно переписувати стандартні функції, такі як меню. Проблема з наборами інструментів полягає в тому, що стилі взаємодії обмежені тими, що надаються. Наприклад, важко створити один повзунок, який містить два індикатори, що може бути корисним для введення верхньої та нижньої меж діапазону. Крім того, самі набори інструментів часто є дорогими у створенні: «Примітиви в принципі ніколи не здаються складними, але програми, які їх реалізують, напрочуд заплутані» . Ще одна проблема з наборами інструментів полягає в тому, що їх часто важко використовувати, оскільки вони можуть містити сотні процедур, і часто незрозуміло, як використовувати процедури для створення бажаного інтерфейсу.

Як і у випадку з графічним пакетом, інструментарій може бути реалізований або за допомогою віконної системи, або використовуватися нею (див. рис. 2.4). Ранні системи надавали лише мінімальні віджети (наприклад, лише меню), а програми очікували, що вони надаватимуть інші (Рис. 2.4-а). У

Macintosh та Microsoft Windows інструментарій знаходиться на низькому рівні, а інтерфейс користувача менеджера вікон побудовано з його використанням. Перевага цього полягає в тому, що менеджер вікон може використовувати ті ж складні процедури набору інструментів для свого інтерфейсу користувача (Рис. 2.4-b). Під час розробки системи X розробники не могли домовитися про єдиний набір інструментів, тому вони залишили його поверх віконної системи. У X програмісти можуть використовувати різноманітні інструментальні набори (наприклад, інструментальні набори Motif, InterViews, Amulet, tcl/tk та Gnome GTK+ можна використовувати поверх X), але менеджер вікон зазвичай повинен реалізовувати свій інтерфейс користувача без використання цього інструментарію (рис. 2.4-c). Інструментальний набір Java “Swing” реалізовано поверх пакета 2D-графіки Java, який, у свою чергу, знаходиться поверх системи керування вікнами (рис. 2.4-d).

Оскільки розробники X не могли домовитися про єдиний зовнішній вигляд та функціональність, вони створили внутрішній шар для побудови різних наборів віджетів, який вони назвали Xt . Цей шар надає загальні послуги, такі як методи об'єктно-орієнтованого програмування та керування макетом. Шар набору віджетів - це колекція віджетів, реалізована за допомогою внутрішніх елементів. Кілька наборів віджетів з різним зовнішнім виглядом та функціональністю можуть бути реалізовані поверх одного й того ж внутрішнього шару (Рисунок 2.4-a), або ж той самий зовнішній вигляд та функціональність може бути реалізований поверх різних внутрішніх елементів (Рис. 2.4-b).

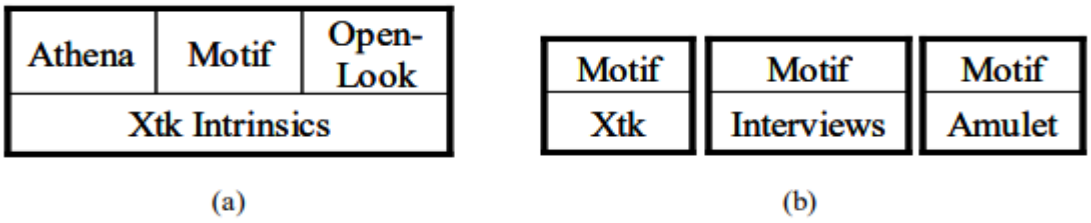

Рисунок 2.4 - (a) Щонайменше три різні набори віджетів, які мають різний вигляд, було реалізовано на основі інтринсиків Xt. (b) Зовнішній вигляд

Motif було реалізовано на багатьох різних інтринсиках.

Внутрішні характеристики інструментарію

Інструментальні набори бувають двох основних різновидів. Найбільш поширений — це просто набір процедур, які можуть викликатися прикладними програмами. Прикладами такого стилю є інструментарій SunTools для віконної системи SunView та оригінальний Macintosh Toolbox. Другий різновид використовує об'єктно-орієнтований стиль програмування, який спрощує для розробника налаштування методів взаємодії. Прикладами є Smalltalk, Andrew, InterViews, Xt, Amulet, інструментарій Java Swing та GTK+ від Gnome.

Переваги використання об'єктно-орієнтованих вбудованих функцій полягають у тому, що це природний спосіб мислення про віджети (меню та кнопки на екрані виглядають як об'єкти), об'єкти віджетів можуть виконувати деякі завдання, які в іншому випадку були б покладені на програміста (наприклад, оновлення), а також легше створювати власні віджети (шляхом створення підкласів існуючого віджета). Перевага старішого, процедурного стилю полягає в тому, що його легше реалізувати, не потрібна спеціальна об'єктно-орієнтована система та легше взаємодіяти з кількома мовами програмування. Для реалізації об'єктів інструментарій може створити власну об'єктну систему, як це було зроблено з Xt, Andrew, Amulet та GTK+, або ж використовувати існуючу об'єктну систему, як це було зроблено в InterViews , який використовує C++, NeXTStep від NeXT, який використовує Objective-C, та Swing, який використовує Java.

Звичайний спосіб взаємодії об'єктно-орієнтованих інструментальних наборів із прикладними програмами полягає у використанні процедур зворотного виклику. Це процедури, визначені програмістом прикладної програми, які викликаються, коли кінцевий користувач працює з віджетом. Наприклад, програміст може надати процедуру, яка викликається, коли користувач вибирає пункт меню. Досвід показав, що реальні інтерфейси часто містять сотні зворотних викликів, що ускладнює модифікацію та підтримку коду . Крім того, різні інструментальні набори, навіть реалізовані на одних і тих самих

вбудованих компонентах, таких як Motif та OpenLook, мають різні ¬протоколи зворотного виклику. Це означає, що код для одного інструментального набору важко перенести на інший. Тому дослідження були спрямовані на зменшення кількості зворотних викликів у програмному забезпеченні інтерфейсу користувача .

Звичайний спосіб взаємодії об'єктно-орієнтованих інструментальних наборів із прикладними програмами полягає у використанні процедур зворотного виклику. Це процедури, визначені програмістом прикладної програми, які викликаються, коли кінцевий користувач працює з віджетом. Наприклад, програміст може надати процедуру, яка викликається, коли користувач вибирає пункт меню. Досвід показав, що реальні інтерфейси часто містять сотні зворотних викликів, що ускладнює модифікацію та підтримку коду . Крім того, різні інструментальні набори, навіть реалізовані на одних і тих самих вбудованих компонентах, таких як Motif та OpenLook, мають різні ¬протоколи зворотного виклику. Це означає, що код для одного інструментального набору важко перенести на інший. Тому дослідження були спрямовані на зменшення кількості зворотних викликів у програмному забезпеченні інтерфейсу користувача .

Зазвичай, внутрішній шар не залежить від зовнішнього вигляду та поведінки, що означає, що віджети, побудовані поверх нього, можуть мати будь-який бажаний зовнішній вигляд та поведінку. Однак, певний набір віджетів повинен вибрати зовнішній вигляд та поведінку. У відео « Усі віджети» показано багато прикладів віджетів, розроблених протягом багатьох років . Наприклад, у ньому показано 35 різних видів меню. Як і інтерфейси користувача менеджера вікон, зовнішній вигляд віджетів може бути захищений авторським правом та запатентований . Як згадувалося вище, різні набори віджетів (з різним зовнішнім виглядом та функціональністю) можуть бути реалізовані поверх тих самих вбудованих функцій. Крім того, той самий зовнішній вигляд та функціональність може бути реалізований поверх різних вбудованих функцій. Наприклад, існують віджети зовнішнього вигляду та функціональності Motif поверх вбудованих

функцій Xt, Interviews та Amulet (Рис. 2.5-b). Хоча всі вони виглядають та працюють однаково (тому кінцевий користувач їх не розрізнить), вони реалізовані зовсім по-різному та мають абсолютно різні процедурні інтерфейси для програміста.

Для підтримки певних типів програм або певних класів програмістів було розроблено низку інструментальних наборів. Наприклад, система SUIT (яка містить інструментарій та конструктор інтерфейсів) спеціально розроблена для легкого вивчення та орієнтована на навчання в класі. Її наступниця, Alice, забезпечує простий спосіб програмування 3D-графіки та анімації. Amulet забезпечує високорівневу підтримку графічних інтерфейсів прямого маніпулювання та обробляє вхідні дані як ієрархічні об'єкти команд, що спрощує реалізацію функції скасування. Rendezvous, Visual Obliq та GroupKit розроблені для спрощення створення програм, що підтримують кількох користувачів на кількох машинах, що працюють синхронно. У той час як більшість інструментальних наборів пропонують лише методи двовимірної взаємодії, набір інструментів Brown 3-D та набір інструментів Inventor від Silicon Graphics надають попередньо запрограмовані 3D-віджети та основу для створення інших. Спеціальну підтримку анімації було додано до Artkit, Amulet та Alice. Tk – це популярний інструментарій для системи X Window (а також Windows), оскільки його можна підключити до різноманітних інтерпретаційних мов, включаючи оригінальну tcl, а також Perl та Python, що дозволяє динамічно змінювати інтерфейс користувача. Tcl підтримує стиль програмування Unix, де багато невеликих програм об'єднані разом.

Хоча між різними наборами інструментів існує багато незначних відмінностей, багато чого залишається незмінним. Наприклад, усі вони мають певний тип меню, кнопки, смуги прокручування, поля введення тексту тощо. Хоча віконних систем та наборів інструментів зараз менше, ніж десять років тому, люди все ще вважають, що перенесення програмного забезпечення так, щоб воно працювало на Macintosh, Microsoft Windows та X на Linux, вимагає багато роботи. Тому було розроблено низку систем, які називаються

віртуальними інструментальниками, що намагаються приховати відмінності між різними інструментальниками, надаючи віртуальні віджети, які можна відобразити на віджети кожного інструментальника. Інша назва цих інструментів - кросплатформні системи розробки. Програміст пише код один раз, використовуючи віртуальний інструментарій, і код працюватиме без змін на різних платформах і все ще виглядатиме так, ніби він був розроблений з віджетами цієї платформи. Наприклад, віртуальний інструментарій може пропонувати одну процедуру меню, яка завжди має той самий інтерфейс програміста, але підключається до меню Motif, меню Macintosh або меню Windows, залежно від того, на якій машині запускається програма.

Існує два стилі віртуальних інструментів. У першому віртуальний інструментарій пов'язаний з різними фактичними інструментами на головному комп'ютері. Наприклад, XVT надає інтерфейс C або C++, який пов'язаний з фактичними інструментами Motif, Macintosh, MS-Windows та OS/2-PM (а також символьними терміналами) та приховує їхні відмінності. Інструментарій Java AWT також працює таким чином і використовує базові набори віджетів. Другий стиль віртуального інструментарію перереалізує віджети в кожному стилі. Наприклад, Galaxy, Amulet та Java Swing надають бібліотеки віджетів, які виглядають як ті, що на різних платформах. Перевагою першого стилю є те, що інтерфейс користувача, швидше за все, буде відповідати зовнішньому вигляду та функціональності (оскільки він використовує реальні віджети). Недоліки полягають у тому, що віртуальний інструментарій все ще повинен надавати інтерфейс до графічних примітивів малювання на платформах. Крім того, вони, як правило, надають лише функції, які з'являються у всіх інструментах. Багато віртуальних інструментальних наборів, що використовують другий підхід, наприклад Java Swing, надають складний графічний пакет та повні набори віджетів на всіх платформах. Однак, при другому підході завжди має бути велика бібліотека виконання, оскільки, окрім вбудованих віджетів, які є рідними для машини, існує повторна реалізація цих самих віджетів у бібліотеці віртуального інструментарію.

Усі набори інструментів, що працюють на кількох платформах, можна вважати віртуальними наборами інструментів другого типу. Наприклад, SUIT та Garnet працюють на X, Macintosh та Windows. Однак вони використовують однаковий зовнішній вигляд на всіх платформах (і тому виглядають інакше, ніж інші програми на цій платформі), тому їх не класифікують як віртуальні набори інструментів.

2.4 Висновки по розділу

Використання інструментів для створення користувацьких інтерфейсів не лише підвищує якість кінцевого продукту, а й значно спрощує процес його розробки, підтримки й масштабування. Це сприяє кращій співпраці між командами та швидшому реагуванню на потреби користувачів.

Інструменти програмного забезпечення для інтерфейсу користувача постійно еволюціонують, спрощуючи розробку складних графічних інтерфейсів і забезпечуючи кращу сумісність, гнучкість та зручність для користувачів. Віконні системи, інструментарії та інструменти вищого рівня відіграють ключову роль у створенні ефективного та адаптивного середовища взаємодії.

Набори інструментів забезпечують стандартизований спосіб створення графічних інтерфейсів, але обмежують гнучкість дизайну і вимагають значних зусиль для міжплатформної сумісності.

РОЗДІЛ 3. ІНСТРУМЕНТИ ТА ПІДХОДИ ДО ПРОЄКТУВАННЯ ІНТЕРФЕЙСУ КОРИСТУВАЧА

3.1 Інструменти вищого рівня

Оскільки програмування на рівні інструментарію є досить складним, існує величезний інтерес до інструментів вищого рівня, які спростять процес створення програмного забезпечення для користувацького інтерфейсу. Вони обговорюються далі.

Багато інструментів вищого рівня мають компоненти, які працюють у різний час. Компонент часу проектування допомагає дизайнеру інтерфейсу користувача розробляти інтерфейс користувача. Наприклад, це може бути графічний редактор, який може розміщувати інтерфейс, або компілятор для обробки мови специфікації інтерфейсу користувача. Наступний етап — це коли кінцевий користувач використовує програму. Тут використовується компонент часу виконання інструменту. Зазвичай він включає інструментарій, але може також включати додаткове програмне забезпечення, спеціально розроблене для цього інструменту. Оскільки компонент часу виконання «керує» інтерфейсом користувача, термін « Система керування інтерфейсом користувача» (UIMS) видається доречним для інструментів зі значним компонентом часу виконання. Також може бути компонент після виконання , який допомагає з оцінкою та налагодженням інтерфейсу користувача. На жаль, дуже мало інструментів для роботи з інтерфейсом користувача мають компонент після виконання. Частково це пов'язано з тим, що інструменти, які пробували це робити, такі як MIKE , виявили, що існує дуже мало метрик, які можуть застосовуватися комп'ютерами. Деякі інструменти намагаються оцінити, як люди взаємодіятимуть з інтерфейсами, автоматично створюючи когнітивні моделі з високорівневих описів інтерфейсу користувача. Наприклад, система GLEAN генерує кількісні прогнози продуктивності системи на основі моделі GOMS .

Інструменти високорівневого користувацького інтерфейсу бувають різних форм. Один із важливих способів їх класифікації - це те, як розробник

визначає, яким має бути інтерфейс. Деякі інструменти вимагають від програміста програмування на спеціальній мові, деякі надають платформу програми для керування програмуванням, деякі автоматично генерують інтерфейс з високорівневої моделі або специфікації, а інші дозволяють інтерактивно розробляти інтерфейс. Кожен із цих типів обговорюється нижче. Звичайно, деякі інструменти використовують різні методи для визначення різних частин користувацького інтерфейсу. Вони класифікуються за їхньою переважною або найцікавішою особливістю.

У більшості старіших інструментів користувацького інтерфейсу розробник визначає інтерфейс користувача спеціальною мовою. Ця мова може приймати різні форми, включаючи контекстно-вільні граматики, діаграми переходів станів, декларативні мови, мови подій тощо. Мова зазвичай використовується для визначення синтаксису користувацького інтерфейсу, тобто допустимих послідовностей вхідних та вихідних дій. Це іноді називають діалогом. Грін надає розширене порівняння граматик, діаграм переходів станів та мов подій, а Олсен розглядає різні старі методи UIMS.

Оскільки багато частин інтерфейсів користувача передбачають обробку послідовності вхідних подій, природно подумати про використання мережі переходів станів для кодування інтерфейсу. Мережа переходів складається з набору станів, з дугами з кожного стану, позначеними вхідними токенами, які спричиняють перехід до стану на іншому кінці дуги. На додаток до вхідних токенів, у деяких системах на дугах також можуть бути розміщені виклики процедур програми та вивід для відображення. У 1968 році Ньюмен реалізував простий інструмент, використовуючи кінцеві автомати , який обробляв текстове введення. Це, очевидно, був перший інструмент для роботи з інтерфейсом користувача. Багато припущень та методів, що використовуються в сучасних системах, були присутні в інструменті Ньюмена: різні мови для визначення інтерфейсу користувача та семантики (семантичні процедури були закодовані звичайною мовою програмування), аналізатор синтаксису, керований таблицями, та незалежність від пристрою .

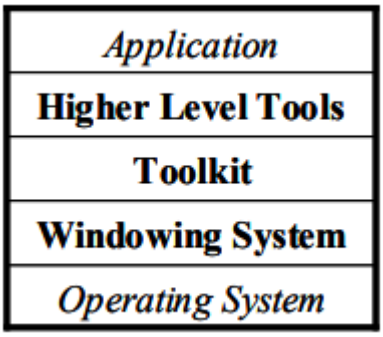

Рисунок 3.1 - Компоненти інтерфейсу користувача програмного забезпечення.

Інструменти для діаграм станів найбільш корисні для створення користувацьких інтерфейсів, де інтерфейс користувача має велику кількість режимів (кожен стан насправді є режимом). Наприклад, діаграми станів корисні для опису роботи низькорівневих віджетів (наприклад, як працює меню або смуга прокручування) або загального глобального потоку програми (наприклад, ця команда відкриє діалогове вікно, з якого можна перейти до цих двох діалогових вікон, а потім до цього іншого вікна тощо). Однак більшість високоінтерактивних систем намагаються бути здебільшого «вільними від режимів», що означає, що в кожній точці користувач має широкий вибір дій. Це вимагає великої кількості дуг з кожного стану, тому інструменти для діаграм станів не були успішними для цих інтерфейсів. Крім того, діаграми станів не можуть обробляти інтерфейси, де користувач може працювати з кількома об'єктами одночасно. Ще одна проблема полягає в тому, що вони можуть бути дуже заплутаними для великих інтерфейсів, оскільки вони стають «лабіринтом проводів», і дуги поза сторінкою (або поза екраном) може бути важко відстежувати.

Визнаючи ці проблеми, але все ще намагаючись зберегти чіткість діаграм переходів станів, Джейкоб винайшов новий формалізм, який являє собою поєднання діаграм станів з певною формою мов подій. Одночасно може бути активним кілька діаграм, а потік керування передається від однієї до іншої у співпроцедурному режимі. Система може створювати різні форми інтерфейсів

прямого маніпулювання. VAPS – це комерційна система, яка використовує модель переходів станів, і вона усуває проблему лабіринту проводів, надаючи таблицю, подібну до електронної таблиці, в якій вказані стани, події та дії . Мережі переходів були ретельно досліджені, але не виявилися особливо успішними чи корисними ні як дослідницький, ні як комерційний підхід.

Багато систем, заснованих на граматиці, базуються на генераторах парсерів, що використовуються в розробці компіляторів. Наприклад, розробник може визначити синтаксис інтерфейсу користувача, використовуючи певну форму BNF. Прикладами систем, заснованих на граматиці, є Syngraph та парсери, побудовані за допомогою YACC та LEX в Unix. Інструменти на основі граматики, такі як інструменти для діаграм станів, не підходять для опису високоінтерактивних інтерфейсів, оскільки вони орієнтовані на пакетну обробку рядків зі складною синтаксичною структурою. Ці системи найкраще підходять для текстових командних мов і здебільшого від них відмовилися дослідники та комерційні розробники для користувацьких інтерфейсів.

У мовах подій вхідні токени вважаються подіями , що надсилаються окремим обробникам подій. Кожен обробник матиме умовний блок, який визначає, які типи подій він оброблятиме та коли він буде активним. Тіло обробника може викликати вихідні події, змінювати внутрішній стан системи (що може вмикати інші обробники подій) або викликати підпрограми програми. Sassafras – це мова програмування подій, де інтерфейс користувача програмується як набір невеликих обробників подій. Мова Elements-Events and Transitions (EET) забезпечує ретельний контроль над тим, коли спрацьовують різні обробники подій . У цих ранніх системах обробники подій були глобальними. У сучасніших системах обробники подій є специфічними для певних об'єктів. Наприклад, мову HyperTalk, яка є частиною HyperCard для Apple Macintosh, можна вважати мовою програмування подій. Visual Basic від Microsoft також містить функції мови програмування подій, оскільки код пишеться для обробки реакції на події об'єктів.

Переваги мов програмування подій полягають у тому, що вони можуть

обробляти кілька активних пристроїв введення одночасно, а також у простій підтримці немодальних інтерфейсів, де користувач може працювати з будь-яким віджетом або об'єктом. Основним недоліком є те, що створити правильний код може бути дуже важко, особливо коли система стає більшою, оскільки потік керування не локалізований, і невеликі зміни в одній частині можуть впливати на багато різних частин програми. Також розробнику зазвичай важко зрозуміти код, коли він досягає нетривіального розміру. Однак успіх HyperTalk, Visual Basic та подібних інструментів показує, що цей підхід підходить для програм малого та середнього розміру. Стиль програмування, що використовується Java Swing та пов'язаними системами, де програміст перевизначає методи, що викликаються, коли відбуваються події, схожий на стиль програмування подій.

Інший підхід полягає у спробі визначити мову, яка є декларативною (яка вказує, що має статися), а не процедурною (як це зробити). Cousin та HP/Apollo's Open-Dialogue дозволяють розробнику визначати інтерфейси користувача таким чином. Підтримувані інтерфейси користувача – це, по суті, форми, де поля можуть бути текстом, який вводиться користувачем, або опціями, вибраними за допомогою меню чи кнопок. Існують також області графічного виводу, які програма може використовувати будь-яким бажаним чином. Програма програми підключена до інтерфейсу користувача через змінні, які можуть бути встановлені та доступні обома сторонами. Оскільки дослідники розширили цю ідею для підтримки складніших взаємодій, специфікація переросла в повноцінні «моделі» програми, а новіші системи описані в розділі «Автоматична генерація на основі моделей». Мови опису макетів, що постачаються з багатьма інструментальними наборами, також є типом декларативної мови.

Наприклад, мова інтерфейсу користувача (UIL) Motif дозволяє визначати розташування віджетів. Оскільки UIL інтерпретується під час запуску програми, користувачі можуть (теоретично) редагувати код UIL для налаштування інтерфейсу. Однак UIL не є повноцінною мовою в тому сенсі, що дизайнер все ще повинен писати код на C для багатьох частин інтерфейсу, включаючи будь-які області, що містять динамічну графіку, та будь-які віджети, що змінюються.

Перевага використання декларативних мов полягає в тому, що розробнику користувацького інтерфейсу не потрібно турбуватися про часову послідовність подій і може зосередитися на інформації, яку потрібно передавати туди-сюди. Недоліком є те, що таким чином можна забезпечити лише певні типи інтерфейсів, а решту потрібно програмувати вручну в «графічних областях», наданих прикладним програмам. Доступні види взаємодій попередньо запрограмовані та фіксовані. Зокрема, ці системи не підтримують такі речі, як перетягування графічних об'єктів, гумові лінії, малювання нових графічних об'єктів або навіть динамічна зміна елементів меню залежно від режиму або контексту програми. Однак ці мови використовувалися як проміжні мови, що описують розташування віджетів (таких як UIL), що генеруються інтерактивними інструментами.

Низка інструментів для роботи з інтерфейсом користувача дозволяє програмісту використовувати обмеження для визначення інтерфейсу користувача . Ранні системи обмежень включають Sketchpad , який був піонером у використанні графічних обмежень у редакторі малюнків, та Thinglab , який використовував обмеження для графічного моделювання. Згодом Thinglab був розширений для допомоги у створенні інтерфейсів користувача . Вищезазначене обговорення інструментальних наборів згадувало використання обмежень як частину внутрішньої структури інструментального набору. Низка дослідницьких інструментальних наборів зараз надають обмеження як невід'ємну частину об'єктної системи (наприклад, Garnet, Amulet та SubArctic). Крім того, деякі системи надають інтерфейси вищого рівня для обмежень. Graphical Thinglab дозволяє розробнику створювати обмеження, об'єднуючи значки разом, а NoPump та C32 дозволяють визначати обмеження за допомогою інтерфейсу, подібного до електронної таблиці. Перевага обмежень полягає в тому, що вони є природним способом вираження багатьох видів зв'язків, які часто виникають у користувацьких інтерфейсах. Наприклад, лінії повинні залишатися прикріпленими до блоків, мітки повинні залишатися по центру всередині блоків тощо. Недоліком обмежень є те, що для їх ефективного розв'язання потрібна

складна система виконання, і програмістам може бути важко правильно вказати та налагодити системи обмежень. Наразі немає комерційних інструментів для користувацького інтерфейсу, що використовують універсальні розв'язувачі обмежень.

Дуже важливий клас комерційних інструментів підтримує доступ до баз даних на основі форм або графічного інтерфейсу. Великі постачальники баз даних, такі як Oracle, надають інструменти, що дозволяють розробникам визначати інтерфейс користувача для доступу та налаштування даних. Часто ці інструменти включають інтерактивні редактори форм (які по суті є конструкторами інтерфейсів) та спеціальні мови баз даних. Мови четвертого покоління (4GL), які підтримують визначення інтерактивних форм для доступу та введення даних, також належать до цієї категорії.

Візуальні програми використовують графіку та дво (або більше) вимірне макетування як частину специфікації програми . Було досліджено багато різних підходів до використання візуального програмування для визначення користувацьких інтерфейсів. Більшість систем, що підтримують мережі переходу станів, використовують візуальне представлення. Іншим популярним методом є використання мов потоку даних. У них значки представляють кроки обробки та потік даних вздовж з'єднувальних проводів. Користувацький інтерфейс зазвичай будується безпосередньо шляхом розміщення попередньо створених віджетів у стилі конструкторів інтерфейсів. Прикладами систем візуального програмування для створення користувацьких інтерфейсів є Labview , яка спеціалізується на управлінні лабораторними приладами, та Prograph . Використання візуальної мови, здається, полегшує роботу програмістам-початківцям, але великі програми все ще страждають від знайомої проблеми «лабіринту проводів». В інших статтях (наприклад) детально проаналізовано сильні та слабкі сторони візуального програмування. Ще однією популярною мовою програмування є Visual Basic від Microsoft. Однак це радше структурний редактор для Basic, поєднаний з конструктором інтерфейсів (див. нижче), і тому насправді не вважається візуальною мовою.

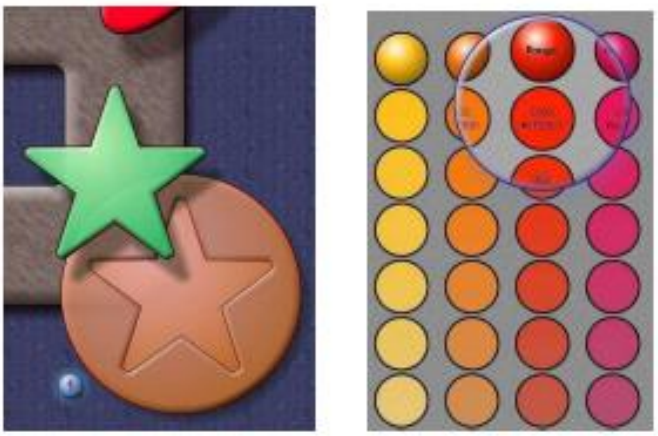

Рисунок 3.2 - Ієрархічні групи, прості перетворення,відсікання, операції з фігурами та градієнти допомагають формувати світло сцени та надають їй більшої виразності

Підсумовуючи, було розроблено багато різних типів мов для опису користувацьких інтерфейсів. Одна з проблем усіх цих мов полягає в тому, що їх можуть використовувати лише професійні програмісти. Деякі програмісти заперечують проти вимоги вивчення нової мови для програмування лише частини користувацького інтерфейс. Це було підтверджено дослідженням ринку. Крім того, здається більш природним визначати графічну частину користувацького інтерфейсу за допомогою графічного редактора. Однак очевидно, що в найближчому майбутньому значну частину користувацького інтерфейсу все ще потрібно буде створювати шляхом написання програм, тому доречно продовжувати дослідження найкращої мови для цього. Дійсно, ціла книга присвячена дослідженню мов для програмування користувацьких інтерфейсів .

Після того, як Macintosh Toolbox став доступним деякий час, Apple виявила, що програмістам важко зрозуміти, як викликати різні функції інструментарію та як забезпечити відповідність отриманого інтерфейсу рекомендаціям Apple. Тому вони створили програмну систему, яка забезпечує загальну основу програми для керівництва програмістів. Вона називалася MacApp і використовувала об'єктно-орієнтовану мову Object Pascal. Класи надаються для важливих частин програми, таких як головні вікна, команди тощо,

і програміст спеціалізує ці класи для надання специфічних для програми деталей, таких як те, що насправді відображається у вікнах і які команди надаються. MacApp дуже успішно спростив написання програм для Macintosh. Сьогодні існує безліч фреймворків, які допомагають створювати програми для більшості основних платформ, включаючи Microsoft Foundation Classes (MFC) для Windows та портативний фреймворк Java Swing. Фреймворк можна описати як архітектуру програмного забезпечення, часто об'єктно-орієнтовану, яка керує програмістом, щоб полегшити реалізацію програмного забезпечення інтерфейсу користувача.

Фреймворк Amulet орієнтований на графічні програми, але завдяки своїй графічній моделі даних багато вбудованих процедур можна використовувати без змін (програмісту зазвичай не потрібно писати методи для підкласів). Новіші фреймворки спрямовані на допомогу в реалізації програм, що використовують переваги повсюдних обчислень (також звані всепроникними обчисленнями), кількох користувачів (також звані комп'ютерно-підтримуваною кооперативною роботою — CSCW), різних датчиків, які повідомляють комп'ютеру, де він знаходиться і хто поруч (так звані контекстно-залежні обчислення), та інтерфейсів користувача, що охоплюють кілька комп'ютерів (так звані багатокомп'ютерні інтерфейси користувача). Наприклад, фреймворк BEACH надає засоби для обробки всіх цих типів інтерфейсів користувача.

Компонентний підхід має на меті замінити сучасні великі монолітні програми меншими частинами, які об'єднуються. Наприклад, ви можете придбати окремий текстовий редактор, лінійку, форматувальник абзаців, перевірку орфографії та програму для малювання, і всі вони працюватимуть разом безперебійно. Цей підхід був винайдений середовищем Andrew , яке забезпечує об'єктно-орієнтовану модель документа, що підтримує вбудовування різних видів даних в інші документи. Ці «вставки» відрізняються від даних, які вирізаються та вставляються в таких системах, як Macintosh, оскільки вони містять програми, які їх редагують, і тому їх завжди можна редагувати на місці. Крім того, документ-контейнер не повинен знати, як відображати або друкувати

вставлені дані, оскільки оригінальна програма, яка їх створила, завжди доступна. Розробник, який створює нову вставку, пише підкласи, що дотримуються стандартного протоколу, щоб система знала, як передавати вхідні події до відповідного редактора. Microsoft OLE, OpenDoc від Apple та JavaBeans використовують цей підхід. Ініціатива Microsoft «.Net» забезпечує компонентну архітектуру для веб-сервісів. Усі ці фреймворки вимагають від розробника написання коду, зазвичай шляхом створення специфічних для програми підкласів стандартних класів, що надаються як частина фреймворку.

3.2 Автоматична генерація на основі моделі

Проблема всіх мовних інструментів полягає в тому, що дизайнер повинен багато вказувати щодо розміщення, формату та дизайну користувацьких інтерфейсів. Щоб вирішити цю проблему, деякі інструменти використовують автоматичну генерацію, завдяки чому інструмент робить багато з цих варіантів на основі специфікації набагато вищого рівня. Багато з цих інструментів, такі як Mickey, Jade та DON , зосереджені на створенні меню та діалогових вікон. Jade дозволяє дизайнеру використовувати графічний редактор для редагування згенерованого інтерфейсу, якщо він недостатньо хороший. DON має найскладніші механізми компонування та враховує бажаний розмір вікна, баланс, симетрію, групування тощо. Автоматичне створення діалогових вікон було дуже ретельно досліджено, але досі немає комерційних інструментів, які б це робили. UIDE (середовище проектування користувацького інтерфейсу) вимагає, щоб семантика програми була визначена спеціальною мовою, і тому може бути включена до інструментів на основі мови. Натомість вона розміщена тут, оскільки мова використовується для опису функцій, які підтримує програма, а не бажаного інтерфейсу. UIDE класифікується як модельно-орієнтований підхід, оскільки специфікація слугує високорівневою, складною моделлю семантики програми. В UIDE опис включає попередні та пост-умови операцій, і система використовує їх для міркування про операції, автоматичної генерації

інтерфейсу та автоматичної генерації довідки .

Система ITS також використовує правила для створення інтерфейсу. ITS використовувалася для створення інформаційної системи для відвідувачів Всесвітньої виставки EXPO 1992 у Севільї, Іспанія. На відміну від інших систем, заснованих на правилах, від дизайнера, який використовує ITS, очікується написання багатьох правил, а не просто написання специфікації, на якій працюють ці правила. Зокрема, філософія дизайну ITS полягає в тому, що всі дизайнерські рішення повинні бути кодифіковані як правила, щоб їх могли використовувати наступні дизайнери, що, сподіваємося, означатиме, що дизайн інтерфейсів стане простішим і кращим, оскільки буде введено більше правил. Як результат, дизайнер ніколи не повинен використовувати графічне редагування для покращення дизайну, оскільки тоді система не зможе виявити причину, чому згенерований дизайн був недостатнім. Останнім часом спостерігається відродження інтересу до інтерфейсів на основі моделей, щоб спробувати забезпечити інтерфейси, які працюють на багатьох видах портативних пристроїв. Наприклад, протокол бездротового доступу (WAP) містить високорівневі описи інформації, що відображатиметься, яку портативні пристрої повинні перетворити для використання певних макетів та методів взаємодії. Тривають дослідження способів перетворення високорівневих специфікацій побутової техніки та інших пристроїв у відповідні інтерфейси дистанційного керування для портативних пристроїв, для використання в «розумних кімнатах», людьми з інвалідністю та для побутової техніки .

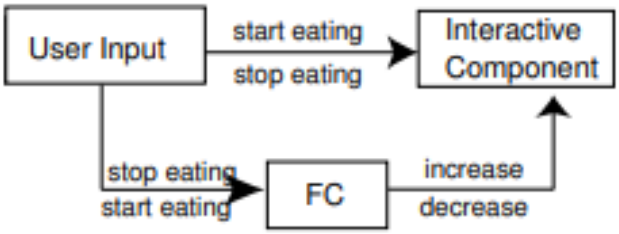

Рисунок 3.3 - Потік подій

Інструменти, описані далі, дозволяють визначити інтерфейс користувача, принаймні частково, шляхом розміщення об'єктів на екрані за допомогою вказівного пристрою. Це мотивовано спостереженням, що візуальне представлення інтерфейсу користувача має першорядне значення в графічних інтерфейсах користувача, і графічний інструмент здається найбільш підходящим способом визначення графічного вигляду. Ще однією перевагою цього методу є те, що він зазвичай набагато простіший у використанні для дизайнера. Багато з цих систем можуть використовуватися навіть непрограмістами. Тому психологи, графічні дизайнери та фахівці з інтерфейсів користувача можуть легше залучатися до процесу проектування інтерфейсу користувача, коли використовуються ці інструменти. Ці інструменти можна відрізнити від тих, що використовують візуальне програмування, оскільки при прямій графічній специфікації фактичний інтерфейс користувача (або його частина) малюється, а не генерується опосередковано з візуальної програми. Таким чином, інструменти прямої графічної специфікації називають програмуванням прямої маніпуляції, оскільки користувач безпосередньо маніпулює віджетами інтерфейсу користувача та іншими елементами. Інструменти, що підтримують графічну специфікацію, можна розділити на чотири категорії: інструменти для створення прототипів, ті, що підтримують послідовність карт, конструктори інтерфейсів та редактори для графіки, специфічної для додатків.

Інструменти для створення прототипів

Мета інструментів для створення прототипів полягає в тому, щоб дозволити дизайнеру швидко створити кілька прикладів того, як виглядатимуть екрани в програмі. Іноді ці інструменти не можна використовувати для створення реального користувацького інтерфейсу програми; вони просто показують, як виглядатимуть деякі аспекти. Це головний фактор, який відрізняє їх від інших високорівневих інструментів. Багато частин інтерфейсу можуть бути непрацездатними, а деякі речі, які виглядають як віджети, можуть бути просто статичними зображеннями. У багатьох прототипувальниках не використовуються справжні віджети інструментарію, а це означає, що дизайнеру

доводиться малювати симуляції, які виглядають як віджети, що з'являться в інтерфейсі. Зазвичай дизайнер витрачає кілька днів або тижнів на випробування різних дизайнів за допомогою інструменту, а потім повністю переробляє остаточний дизайн в окремій системі. Більшість інструментів для створення прототипів можна використовувати без програмування, тому їх, наприклад, можуть використовувати графічні дизайнери.

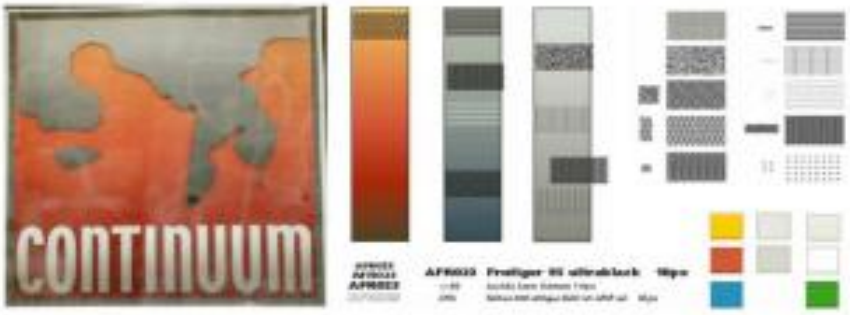

Рисунок 3.4 - Робота дизайнера над «глобальною картиною»

Зверніть увагу, що таке використання терміна «прототипування» відрізняється від загального терміну «швидке прототипування», який став модним маркетинговим терміном. У рекламі майже всіх інструментів для роботи з інтерфейсом користувача стверджується, що вони підтримують «швидке прототипування», під чим вони мають на увазі, що інструмент допомагає швидше створювати програмне забезпечення інтерфейсу користувача. Термін «прототипування» використовується в цій статті набагато конкретніше. Ймовірно, першим інструментом для створення прототипів була програма Demo Дена Брікліна. Це програма для IBM PC, яка дозволяє дизайнеру створювати зразки екранів, що складаються із символів та символьної графіки (де комірки символів фіксованого розміру можуть містити графіку, таку як горизонтальна, вертикальна або діагональна лінія). Дизайнер може легко створювати різні екрани для програми. Також відносно легко вказати дії (миша або клавіатура), які викликають переходи з одного екрана на інший. Однак важко визначити інші поведінкові параметри. Загалом, в інструментах для створення прототипів може бути певна підтримка полів введення та меню, але можливості для обробки або

тестування результатів обмежені.

Для графічних інтерфейсів користувача дизайнери часто використовують такі інструменти, як Director від Macromedia , який насправді є інструментом анімації. Дизайнер може намалювати приклади екранів, а потім вказати, що при натисканні миші в певному місці має запускатися анімація або відображатися інший екран. Компоненти зображення можна повторно використовувати на різних екранах, але знову ж таки, можливості відображення поведінки обмежені. HyperCard та Visual Basic також часто використовуються як інструменти для створення прототипів. Дослідницькі інструменти, такі як SILK та DENIM , забезпечують швидкий інтерфейс для створення ескізів, а потім перетворюють ескізи на фактичні інтерфейси. Основним недоліком цих інструментів для створення прототипів є те, що іноді застосунок необхідно перекодувати «реальною» мовою, перш ніж його буде надано. Також існує ризик того, що програмісти, які реалізують реальний користувацький інтерфейс, проігнорують прототип.

Карткиж. Багато графічних програм обмежені користувацькими інтерфейсами, які можна представити як послідовність переважно статичних сторінок, які іноді називають фреймами, картками або формами. Кожна сторінка містить набір віджетів, деякі з яких здійснюють перехід на інші сторінки. Зазвичай існує фіксований набір віджетів на вибір, які були написані вручну. Раннім прикладом цього є Menulay , який дозволяє дизайнеру розміщувати текст, графічні потенціометри, значки та світлові кнопки на екрані та бачити саме те, що побачить кінцевий користувач під час запуску програми. Дизайнеру не потрібно бути програмістом, щоб використовувати Menulay. Ймовірно, найвідомішим прикладом системи на основі карток є HyperCard від Apple. Існує багато подібних програм, таких як GUIDE та ToolBook . У всіх них дизайнер може легко створювати картки, що містять текстові поля, кнопки тощо, а також різні графічні елементи оформлення. Кнопки здійснюють перенесення на інші картки. Ці програми надають мову сценаріїв для забезпечення більшої гнучкості для кнопок. Мова сценаріїв HyperCard називається HyperTalk і, як згадувалося

вище, насправді є мовою подій, оскільки програміст пише короткі фрагменти коду, які виконуються, коли відбуваються події введення. У своєму первісному вигляді Всесвітня павутина була представлена як послідовність сторінок, які були схожі на «картки» з вбудованими посиланнями, що переносять на інші сторінки, що замінюють попередню сторінку.

Конструктори інтерфейсів дозволяє дизайнеру створювати діалогові вікна, меню та вікна, які мають бути частиною більшого користувацького інтерфейсу. Їх також називають інструментами розробки інтерфейсів (IDT) або конструкторами графічного інтерфейсу. Конструктори інтерфейсів дозволяють дизайнеру вибирати з попередньо визначеної бібліотеки віджетів та розміщувати їх на екрані за допомогою миші. Інші властивості віджетів можна налаштувати за допомогою аркушів властивостей. Зазвичай також є певна підтримка послідовності, наприклад, виклик піддіалогів при натисканні певної кнопки. Проект Steamer у BBN продемонстрував багато ідей, які пізніше були включені в конструктори інтерфейсів, і, ймовірно, був першою об'єктно-орієнтованою графічною системою. Іншими прикладами дослідницьких конструкторів інтерфейсів є DialogEditor та Gilt . Існують сотні комерційних конструкторів інтерфейсів, включаючи редактори ресурсів , що постачаються з професійними середовищами розробки, такими як Metrowerks Codewarrior. Visual Basic від Microsoft, по суті, є конструктором інтерфейсів у поєднанні з редактором для інтерпретованої мови. Багато з обговорених вище інструментів, таких як віртуальні набори інструментів, візуальні мови та фреймворки додатків, також містять конструктори інтерфейсів.

Розробники інтерфейсів використовують фактичні віджети з інструментарію, тому їх можна використовувати для створення частин реальних програм. Більшість із них генерують шаблони коду на C або C++, які можна компілювати разом із кодом програми. Інші генерують опис інтерфейсу мовою, яку можна читати під час виконання. Іноді важливо, щоб програмісти не редагували вивід інструментів (наприклад, згенерований код C), інакше інструмент більше не можна буде використовувати для подальших модифікацій.

Хоча конструктори інтерфейсів спрощують компонування діалогових вікон та меню, це лише частина проблеми дизайну інтерфейсу користувача. Ці інструменти надають мало орієнтирів на створення хороших інтерфейсів користувача, оскільки вони надають дизайнерам значну свободу. Ще одна проблема полягає в тому, що для будь-якої програми, яка має графічну область (наприклад, програми для малювання, САПР, редактори візуальної мови тощо), конструктори інтерфейсів не... допомога з вмістом графічної панелі. Також вони не можуть обробляти віджети, які змінюються динамічно. Наприклад, якщо вміст меню або макет діалогового вікна змінюється залежно від стану програми, це потрібно запрограмувати шляхом написання коду.

Інструменти візуалізації даних

Важливою комерційною категорією інструментів є системи динамічної візуалізації даних. Ці інструменти, які, як правило, досить дорогі, зосереджуються на відображенні динамічно змінних даних на комп'ютері та використовуються як інтерфейси для моделювання, керування процесами, моніторингу систем, управління мережею та аналізу даних. Інтерфейс дизайнера зазвичай дуже схожий на конструктор інтерфейсів, з палітрою датчиків, графічних інструментів, ручок та перемикачів, які можна розміщувати інтерактивно. Однак ці елементи керування зазвичай не входять до набору інструментів, а надаються самим інструментом. Прикладами інструментів цієї категорії є DataViews та SL-GMS .

Редактори для графіки, спеціалізованої для додатків

Коли програма має власну графіку, було б корисно, якби дизайнер міг малювати зображення того, як має виглядати графіка, замість того, щоб писати для цього код. Проблема полягає в тому, що графічні об'єкти зазвичай потрібно змінювати під час виконання, виходячи з фактичних даних та дій кінцевого користувача. Тому дизайнер може намалювати лише приклад бажаного відображення, який буде змінено під час виконання, тому ці інструменти називаються демонстраційним програмуванням. Це відрізняє ці програми від графічних інструментів попередніх трьох розділів, де повне зображення можна

визначити під час проектування. В результаті завдання узагальнення перетворення прикладів об'єктів у параметризовані прототипи, які можуть змінюватися під час виконання, ці системи все ще перебувають на стадії дослідження.

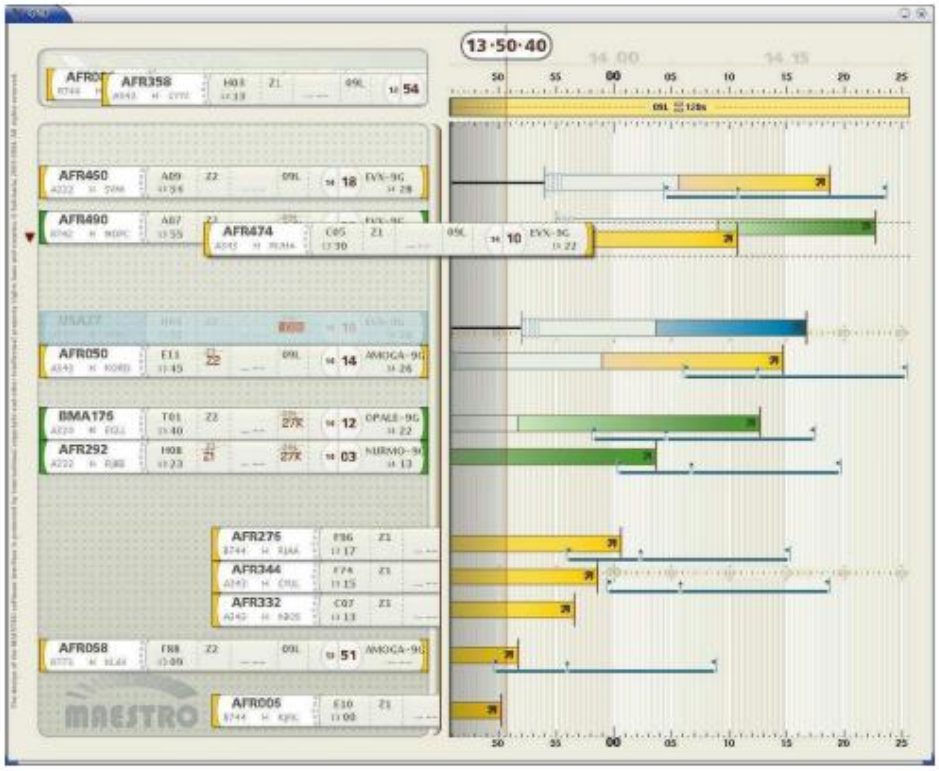

Рисунок 3.5 - Фінальний додаток після повної інтеграції

Peridot дозволяє створювати нові, користувацькі віджети. Примітиви, якими дизайнер маніпулює за допомогою миші, - це прямокутники, кола, текст і лінії. Система узагальнює дії дизайнера для створення параметризованих, об'єктно-орієнтованих процедур, таких як ті, що можна знайти в інструментаріях. Експерименти показали, що Peridot можуть використовувати люди, які не є програмістами. Lapidary розширює ідеї Peridot, дозволяючи малювати загальні об'єкти, специфічні для певного застосування. Наприклад, дизайнер може малювати вузли та дуги для графічної програми. Система DEMO дозволяє демонструвати деякі динамічні властивості об'єктів під час виконання, такі як те, як створюються об'єкти. Інструмент Marquise дозволяє дизайнеру демонструвати, коли повинні відбуватися різні поведінки, і підтримує палітри, які керують цією поведінкою. За допомогою Pavlov користувач може

продемонструвати, як віджети повинні керувати рухом автомобіля в грі з водіння. У Gamut користувач може давати підказки, які допоможуть системі виводити складні моделі поведінки для ігрових застосунків. Дослідження щодо практичного втілення цих ідей тривають.

Спеціалізовані інструменти

Для деяких прикладних областей існують спеціалізовані інструменти, які забезпечують значну підтримку високого рівня. Однак вони, як правило, досить дорогі (тобто від 20 000 до 50 000 доларів США). Наприклад, у сферах аеронавтики та управління в реальному часі існує низка високорівневих інструментів, таких як InterMAPhics .

3.3 Інструменти для Всесвітньої павутини

Реалізація користувацьких інтерфейсів для Всесвітньої павутини зазвичай використовує зовсім інші інструменти, ніж створення графічних інтерфейсів, і детально розглядається в інших розділах цього тому. Крім того, технології та інструменти змінюються досить швидко. Тому цей розділ містить лише короткий огляд. Прості веб-сторінки можуть складатися зі статичного тексту та графіки з вбудованими посиланнями, і їх можна створювати шляхом безпосереднього написання базового html-коду (мова гіпертекстової розмітки). Крім того, автор може використовувати інтерактивні інструменти, такі як Microsoft FrontPage, який таким чином служить своєрідним конструктором інтерфейсів. FrontPage також може створювати сторінки, що містять форми для заповнення інформації (текстові поля, кнопки тощо). Більш динамічні сторінки можуть використовувати мову сценаріїв, вбудовану в html, таку як Javascript або VBscript (Visual Basic Script). Як альтернатива, можна використовувати спеціалізовану мову анімації, таку як мова Flash від Macromedia, яку можна створити за допомогою інтерактивного інструменту, такого як Dreamweaver .

У всіх випадках серверна частина, яка надає сторінки, обробляє будь-які

дані, введені в поля форми, та в результаті надає нові сторінки, має бути реалізована за допомогою певного інструменту серверного скриптингу або роботи з базами даних, який зазвичай досить сильно відрізняється від інструментів, що використовуються для створення клієнтських сторінок, які бачить користувач.

Оскільки створення програмного забезпечення для користувацького інтерфейсу є настільки складним завданням, було зроблено низку спроб описати організацію програмного забезпечення на дуже абстрактному рівні, шляхом створення моделей програмного забезпечення. Найперші спроби використовували ті ж рівні, що були визначені для компіляторів, і говорили про семантичну , синтаксичну та лексичну частини користувацького інтерфейсу, але це виявилося корисним здебільшого лише для реалізацій на основі парсерів . Ще однією ранньою моделлю є модель Зеєхайма , яка відокремлює аспекти представлення (вихід) від управління діалогами (що відбувається в якому порядку, на основі яких вхідних даних) від моделі інтерфейсу програми (які результуючі зміни в даних). Ця модель не дуже добре працює з графічними інтерфейсами, оскільки вони применшують значення діалогу на користь взаємодії без режимів.

Оскільки створення програмного забезпечення для користувацького інтерфейсу є настільки складним завданням, було зроблено низку спроб описати організацію програмного забезпечення на дуже абстрактному рівні, шляхом створення моделей програмного забезпечення. Найперші спроби використовували ті ж рівні, що були визначені для компіляторів, і говорили про семантичну, синтаксичну та лексичну частини користувацького інтерфейсу, але це виявилося корисним здебільшого лише для реалізацій на основі парсерів . Ще однією ранньою моделлю є модель Зеєхайма, яка відокремлює аспекти представлення (вихід) від управління діалогами (що відбувається в якому порядку, на основі яких вхідних даних) від моделі інтерфейсу програми (які результуючі зміни в даних). Ця модель не дуже добре працює з графічними інтерфейсами, оскільки вони применшують значення діалогу на користь

взаємодії без режимів.

Інструменти користувацького інтерфейсу – це галузь, де дослідження мали величезний вплив на сучасну практику розробки програмного забезпечення . Звичайно, менеджери вікон та результуючий «стиль графічного інтерфейсу» походять із фундаментальних досліджень у Стенфордському дослідницькому інституті, Дослідницькому центрі Xerox Palo Alto (PARC) та MIT у 1970-х роках. Конструктори інтерфейсів та «карткові» програми, такі як HyperCard, були винайдені в дослідницьких лабораторіях BBN, Університету Торонто, Xerox PARC та інших. Зараз конструктори інтерфейсів широко використовуються для комерційної розробки програмного забезпечення. Мови подій, які широко використовуються в HyperTalk та Visual Basic, вперше досліджувалися в дослідницьких лабораторіях. Сучасне покоління середовищ, такі як OLE та Java Beans, базується на компонентній архітектурі, яка була розроблена в середовищі Andrew з Університету Карнегі-Меллона. Таким чином, хоча деякі ранні підходи до UIMS, такі як мережі переходів та граматики, могли бути невдалими, загалом дослідження інструментів користувацького інтерфейсу змінили спосіб розробки програмного забезпечення.

Хоча існує багато інструментів для роботи з інтерфейсом користувача, є багато галузей, у яких потрібні подальші дослідження. У попередніх звітах детально обговорюються ідеї майбутніх досліджень інструментів для роботи з інтерфейсом користувача . Тут коротко наведено деякі з важливих.

Нові мови програмування. Вбудовані примітиви введення/виведення в сучасних мовах програмування, такі як printf/scanf або cout/cin, підтримують текстовий стиль запитань і відповідей у інтерфейсі користувача, який є модальним і добре відомий своєю неякісністю. Більшість сучасних інструментів використовують бібліотеки та інтерактивні програми, які є окремими від мов програмування. Однак багато методів, такі як об'єктно-орієнтоване програмування, багатопроцесорна обробка та обмеження, найкраще реалізувати як частину мови програмування. Навіть нові мови, такі як Java ускладнює програмування значної частини інтерфейсу користувача, залишаючи його в

окремих бібліотеках. Крім того, інтегроване середовище, де графічні частини програми можна задати графічно, а решту - текстово, значно спростило б створення програм. Те, як мови програмування можна покращити для кращої підтримки програмного забезпечення інтерфейсу користувача, є темою книги.

Збільшена глибина. Багато дослідників намагаються створити інструменти, які охоплюватимуть більше функцій інтерфейсу користувача, таких як графіка та поведінка, специфічні для додатків. Завдання полягає в тому, щоб забезпечити гнучкість для розробників додатків, водночас забезпечуючи високий рівень підтримки. Інструменти також повинні мати можливість підтримувати довідку, скасування та переривання операцій. Сучасні інструменти для роботи з інтерфейсом користувача здебільшого допомагають у генерації коду інтерфейсу та припускають, що фундаментальний дизайн інтерфейсу користувача завершений. Також необхідні інструменти, які допоможуть у генерації, специфікації та аналізі дизайну інтерфейсу. Наприклад, важливим першим кроком у проектуванні інтерфейсу користувача є аналіз завдань, де дизайнер визначає конкретні завдання, які кінцевий користувач повинен буде виконати. Дослідження слід спрямовувати на створення інструментів для підтримки цих методів і прийомів. Зрештою, вони можуть бути інтегровані з інструментами генерації коду, щоб інформація, згенерована під час раннього проектування, могла бути передана в інструменти автоматичної генерації, можливо, для створення інтерфейсу безпосередньо на основі ранніх аналізів. Інформація також може бути використана для автоматичної генерації документації та довідки під час виконання. Інший підхід полягає в тому, щоб дозволити розробнику визначити дизайн у відповідній нотації, а потім надати інструменти для перетворення цієї нотації в інтерфейси. Наприклад, UAN – це нотація для вираження дій кінцевого користувача та відповідей системи. Зрештою, потрібна велика робота над інструментами, які допоможуть оцінити дизайн інтерфейсів. Початкові спроби, такі як у MIKE , підкреслили необхідність кращих моделей та метрик для оцінки користувацьких інтерфейсів. Дослідження в цій галузі, проведені когнітивними психологами та іншими дослідниками

користувацьких інтерфейсів тривають.

Збільшена широта. Ми можемо очікувати, що інтерфейси користувача завтрашнього дня відрізнятимуться від традиційних інтерфейсів вікон і миші сьогодні, і інструменти повинні будуть змінитися, щоб підтримувати нові стилі. Наприклад, ми вже бачимо крихітні цифрові пейджери та телефони з вбудованими комп'ютерами та дисплеями, комп'ютери розміром з долоню, такі як пристрої PalmOS, панельні комп'ютери розміром з ноутбук, такі як планшетні ПК Microsoft, а також дисплеї розміром з стіну. Крім того, обчислювальна техніка з'являється у все більшій кількості пристроїв вдома та в офісі. Важлива наступна хвиля з'явиться, коли всі пристрої зможуть легко взаємодіяти один з одним, ймовірно, за допомогою бездротових радіотехнологій, таких як 802.11 ("Wi-Fi") або Bluetooth. Звук, відео та анімація будуть все частіше інтегруватися в інтерфейси користувача. Нові пристрої та методи введення, ймовірно, замінять традиційні стилі миші та меню. Наприклад, значно більше будуть використовуватися такі методи, як жести, рукописний текст та введення та виведення мовлення. Вони називаються заснованими на розпізнаванні, оскільки для інтерпретації вхідного потоку від користувача для ідентифікації вмісту потрібне програмне забезпечення. У цих «не-WIMP» програмах (WIMP розшифровується як Windows, Icons, Menus та Pointing Devices), дизайнерам також знадобиться кращий контроль над часом інтерфейсу для підтримки анімації та різних нових медіа, таких як відео. Хоча деякі інструменти орієнтовані на програми для кількох користувачів, немає прямих інструментів для графічної специфікації, а поточні інструменти обмежені в стилях програм, які вони підтримують. Ще однією проблемою є підтримка кількох інтерфейсів для однієї програми, тому вона може працювати на малих і великих пристроях узгоджено, а іноді людина може використовувати кілька пристроїв одночасно , таких як пристрій PalmOS та великий екран дисплея .

Ще однією проблемою є підтримка інтерфейсів, які можна переносити з однієї природної мови на іншу (наприклад, з англійської на французьку). Інтернаціоналізація інтерфейсу набагато складніша, ніж простий переклад

текстових рядків, і включає різні формати чисел, дати та часу, нові методи введення, перероблені макети, різні колірні схеми та нові значки . Як майбутні інструменти можуть допомогти в цьому процесі?

Програмування та налаштування для кінцевого користувача . Однією з найуспішніших комп'ютерних програм усіх часів є електронна таблиця. Основною причиною її успіху є те, що кінцеві користувачі можуть програмувати (записуючи формули та макроси). Однак програмування кінцевим користувачем рідко зустрічається в інших програмах, а там, де воно існує, зазвичай вимагає вивчення традиційного програмування. Наприклад, AutoCAD надає Lisp для налаштування, а багато програм Microsoft використовують Visual Basic. Потрібні ефективніші механізми для користувачів, щоб налаштовувати існуючі програми та створювати нові. Однак їх не слід вбудовувати в окремі програми, як це робиться сьогодні, оскільки це означає, що користувач повинен вивчити різну техніку програмування для кожної програми. Натомість, ці можливості повинні надаватися на системному рівні, і тому вони повинні бути частиною базового інструментарію. Природно, оскільки це орієнтовано на кінцевих користувачів, це не буде схоже на програмування на C, а радше на якомусь вищому рівні.

Розділення застосунків та інтерфейсу користувача. Одна з фундаментальних цілей інструментів для роботи з користувацьким інтерфейсом полягає в тому, щоб забезпечити кращу модуляризацію та відокремлення коду користувацького інтерфейсу від коду програми. Однак, опитування показало, що традиційні інструментальні набори насправді ускладнюють це відокремлення через велику кількість необхідних процедур зворотного виклику. Тому необхідні подальші дослідження способів кращої модуляризації коду та того, як інструменти можуть це підтримувати.

Інструменти для інструментів. Дуже важко створити інструменти, описані в цьому розділі. Кожен з них вимагає величезних зусиль. Тому потрібна робота над тим, щоб зробити створення самих інструментів легшим. Наприклад, інструментарій Garnet досліджував механізми, спеціально розроблені для спрощення створення високорівневих графічних інструментів. Фреймворк

Unidraw також виявився корисним для створення конструкторів інтерфейсів. Однак потрібна подальша робота.

Загалом, дослідження та інновації в інструментах відстають від інновацій у дизайні користувацького інтерфейсу, оскільки розробляти інструменти має сенс лише тоді, коли ви знаєте, для яких типів інтерфейсів ви створюєте інструменти. Враховуючи консолідацію стилю взаємодії з користувацьким інтерфейсом за останні 15 років, не дивно, що інструменти дозріли до такої міри, що комерційні інструменти досить успішно охоплюють важливі аспекти побудови користувацького інтерфейсу. Зрозуміло, що дослідження програмних інструментів користувацького інтерфейсу мали величезний вплив на процес розробки програмного забезпечення. Зараз дизайн користувацького інтерфейсу готовий до радикальних змін, головним чином спричинених розвитком Всесвітньої мережі, повсюдних обчислень, користувацьких інтерфейсів на основі розпізнавання, портативних пристроїв, бездротового зв'язку та інших технологій. Тому ми очікуємо відродження інтересу та досліджень програмних інструментів користувацького інтерфейсу з метою підтримки нових стилів користувацького інтерфейсу.

3.4 Висновки по розділу

Інструменти вищого рівня для створення користувацьких інтерфейсів значно спрощують розробку, пропонуючи різноманітні підходи — від мов програмування подій до візуального та декларативного програмування, хоча жоден з них не є універсальним рішенням.

Інструменти автоматичної генерації інтерфейсів значно спрощують проєктування, однак часто потребують або високорівневої специфікації, або демонстрацій дій, залишаючи виклики в адаптації динамічного чи спеціалізованого контенту.Інструменти для створення користувацьких інтерфейсів відіграють ключову роль у розвитку програмного забезпечення, і з появою нових технологій виникає потреба в їх подальшому вдосконаленні для підтримки інноваційних стилів взаємодії.

ВИСНОВКИ

Дослідження, проведене в рамках цієї роботи, підкреслює ключову роль програмних технологій у проєктуванні інтерфейсів користувача (UI), які забезпечують ефективну взаємодію між користувачами та цифровими системами. Аналіз концептуальних основ показав, що еволюція підходів до проєктування UI, від ранніх командних інтерфейсів до сучасних адаптивних і інтерактивних рішень, відображає зростання складності користувацьких потреб і технологічних можливостей. Модель експертності користувачів проти складності інтерфейсів виявила необхідність балансу між простотою і функціональністю, тоді як вплив досвіду користувача на оцінку зручності використання підкреслив важливість персоналізації та адаптивності в дизайні UI.

Огляд інструментів проєктування UI, представлений у другому розділі, продемонстрував, що сучасні програмні рішення, такі як Figma, Sketch, Adobe XD, а також бібліотеки React і Vue.js, значно спрощують процес створення інтерфейсів. Важливість цих інструментів полягає в їх здатності підтримувати спільну роботу, прототипування та інтеграцію з розробкою. Огляд програмного забезпечення та наборів інструментів показав, що вибір інструменту залежить від специфіки проєкту, рівня автоматизації та потреб команди, що підкреслює необхідність гнучких і модульних рішень.

Автоматична генерація на основі моделей дозволяє скоротити час розробки та підвищити якість інтерфейсів, особливо в умовах швидкого прототипування. Інструменти для веб-розробки, такі як фреймворки JavaScript і CSS, забезпечують створення кросплатформних і адаптивних інтерфейсів, що відповідають сучасним стандартам доступності та продуктивності.

Узагальнюючи, програмні технології проєктування інтерфейсу користувача є критично важливими для створення зручних, естетичних і функціональних цифрових продуктів. Результати дослідження можуть бути використані дизайнерами, розробниками та менеджерами продуктів для вдосконалення процесів проєктування та підвищення якості взаємодії з

користувачами. Перспективи подальших досліджень включають інтеграцію штучного інтелекту для автоматичного дизайну UI, розвиток інструментів для підтримки доступності та вдосконалення технологій для створення іммерсивних інтерфейсів, таких як AR/VR. Ці напрямки сприятимуть розвитку більш інноваційних і користувацько-орієнтованих UI-рішень у майбутньому.

СПИСОК ПОСИЛАНЬ НА ДЖЕРЕЛА

1. Buley, L. The User Experience Team of One: A Research and Design Survival Guide. - Sebastopol, CA: O'Reilly Media, 2013. - 246 с. - Режим доступу: https://www.oreilly.com/library/view/the-user-experience/9781449373405/

2. Cooper, A., Reimann, R., Cronin, D., & Noessel, C. About Face: The Essentials of Interaction Design. - 4th ed. - Indianapolis, IN: Wiley, 2014.- 720 с. - Режим доступу: https://www.wiley.com/en-us/About+Face%3A+The+ Essentials+of+ Interaction+Design%2C+4th+Edition-p-9781118766576

3. Figma Team. Figma Documentation: Design Systems and Prototyping. - 2024. - Режим доступу: https://www.figma.com/developers/docs

4. Garrett, J. J. The Elements of User Experience: User-Centered Design for the Web and Beyond. - 2nd ed. - Berkeley, CA: New Riders, 2010. - 192 с. - Режим доступу: https://www.peachpit.com/store/elements-of-user-experience-user-centered-design-9780321683687

5. Goodwin, K. Designing for the Digital Age: How to Create Human-Centered Products and Services.-Indianapolis, IN: Wiley, 2009. -768 с.-Режим доступу:https://www.wiley.com/en-us/Designing+for+the+Digital+Age-p-9780470229101

6. Hartson, R., & Pyla, P. S. The UX Book: Process and Guidelines for Ensuring a Quality User Experience. -2nd ed.- Cambridge, MA: Morgan Kaufmann, 2018. - 992 с. - Режим доступу: https://www.elsevier.com/books/the-ux-book/hartson/978-0-12-805342-3

7. Hovorushchenko, T., & Alekseiko, V. Aspects of Land Surface Temperature Forecasting for Sustainable Cities. - Хмельницький національний університет, 2024. - С. 1–12. - Режим доступу: https://elar.khmnu.edu.ua

8. Klyuba, M. A**.** Програмні засоби для створення інтерфейсів, що забезпечують взаємодію користувача з Arduino по Bluetooth. - магістерська дисертація. - Київ: НТУУ КПІ, 2024. - 130 с. Режим доступу: https://ela.kpi.ua

9. Krug, S. Don’t Make Me Think, Revisited: A Common Sense Approach to Web Usability. - 3rd ed. - Berkeley, CA: New Riders, 2014. - 216 с. -Режим доступу:https://www.peachpit.com/store/dont-make-me-think-revisited-9780321965 516

10. Lidwell, W., Holden, K., & Butler, J. Universal Principles of Design. - 2nd ed. - Beverly, MA: Rockport Publishers, 2010. - 272 с. - Режим доступу: https://www.quartoknows.com/books/9781592535873/Universal-Principles-of-Design.html

11. Lowdermilk, T. User-Centered Design: A Developer’s Guide to Building User-Friendly Applications. - Sebastopol, CA: O’Reilly Media, 2013. - 154 с. - Режим доступу:https://www.oreilly.com/library/view/user-centered-design/9781449359812/

12. Lysenko, S., & Shpuliar, Y. SVM Classifier Method for Data Analysis on FPGA.-Хмельницький національний університет, 2024. - С. 1–10. - Режим доступу: https://elar.khmnu.edu.ua

13. Marcotte, E. Responsive Web Design. - 2nd ed. - New York, NY: A Book Apart, 2014. - 160 с. - Режим доступу: https://abookapart.com/products/responsive-web-design

14. Morville, P., & Rosenfeld, L. Information Architecture for the World Wide Web. - 4th ed. -Sebastopol, CA: O’Reilly Media, 2015.-528 с.-Режим доступу:https://www.oreilly.com/library/view/information-architecture/9781491911679/

15. Nielsen, J., & Loranger, H. Prioritizing Web Usability. - Berkeley, CA: New Riders, 2006.- 432с.-Режим доступу: https://www. peachpit.com /store/ prioritizing-web-usability-9780321350312

16. Norman, D. A. The Design of Everyday Things. - Revised ed. - New York, NY: Basic Books, 2013.-368с.-Режимдоступу: https://www.basicbooks. com/titles/don-norman/the-design-of-everyday-things/ 9780465050659/

17. Pernice, K. UX Strategy: How to Devise Innovative Digital Products that People Want. - Sebastopol, CA: O’Reilly Media, 2021. - 300 с. - Режим доступу: https://www.oreilly.com/library/view/ux-strategy/9781492050896/

18. Rubin, J., & Chisnell, D. Handbook of Usability Testing: How to Plan, Design, and Conduct Effective Tests. - 2nd ed. - Indianapolis, IN: Wiley, 2008. - 384 с. - Режим доступу: https://www.wiley.com/en-us/Handbook+of+Usability+ Testing -p-9780470185483

19. Shneiderman, B., Plaisant, C., Cohen, M., & Jacobs, **S.** Designing the User Interface: Strategies for Effective Human-Computer Interaction. — 6th ed. — Boston, MA: Pearson, 2016. - 624 с. - Режим доступу: https://www.pearson.com/store/p/designing-the-user-interface/P100000576135

20. Stfalcon Team. Major UI Web Design Principles to Follow in 2024. - 2023. - 10 с. - Режим доступу: https://stfalcon.com/en/blog/post/ui-web-design-principles-2024

21. Tidwell, J., Brewer, C., & Valencia, A. Designing Interfaces: Patterns for Effective Interaction Design. - 3rd ed. - Sebastopol, CA: O'Reilly Media, 2020. - 584 с. - Режим доступу: https://www.oreilly.com/library/view/designing-interfaces/ 9781492051961/

22. Tullis, T., & Albert, B. Measuring the User Experience: Collecting, Analyzing, and Presenting Usability Metrics. - 2nd ed. - Waltham, MA: Morgan Kaufmann,2013.-320 с.-Режим доступу: https://www.elsevier.com/books /measuring-the-user-experience/tullis/978-0-12-415781-1

23. Unger, R., & Chandler**, C.** A Project Guide to UX Design: For User Experience Designers in the Field or in the Making. - 2nd ed. - Berkeley, CA: New Riders, 2012. - 360 с. - Режим доступу: https://www.peachpit.com/store/project-guide-to-ux-design-9780321815385

24. Weinschenk, S. 100 Things Every Designer Needs to Know About People. - Berkeley, CA: New Riders, 2011.-256 с.-Режим доступу: https://www.peachpit.com/store/100-things-every-designer-needs-to-know-about-9780321767530

25. Wroblewski, L. Web Form Design: Filling in the Blanks. - Brooklyn, NY: Rosenfeld Media, 2008. - 226 с. - Режим доступу: https://rosenfeldmedia.com /books/web-form-design/

26. Yablonski, J. Laws of UX: Using Psychology to Design Better Products & Services. - Sebastopol, CA: O'Reilly Media, 2020. -152 с. - Режим доступу: https://www. oreilly.com/library/view/laws-of-ux/9781492055303/

27. Zurb Foundation. Foundation for Sites: Responsive Front-End Framework Documentation. - 2024. - Режим доступу: https://get.foundation /sites/docs/

БІБЛІОГРАФІЧНА ДОВІДКА

Тема бакалаврської роботи: " Програмні технології проектування інтерфейсу користувача"

Обсяг пояснювальної записки: 85 аркушів

Дата закінчення дипломної роботи 15 червня 2024р.

Підпис студента _____