МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ

УКРАЇНСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ ЗАЛІЗНИЧНОГО ТРАНСПОРТУ

С. І. Доценко

ЛЮДИНО-МАШИННИЙ ІНТЕРФЕЙС

Навчальний посібник

Харків 2022

УДК 004.65(075)
Д 714

Рекомендовано вченою радою Українського державного університету залізничного транспорту як навчальний посібник (витяг з протоколу № 4 від 20 квітня 2021 року)

Рецензенти:

професори Г. А. Кучук (НТУ «ХПІ»),
С. О. Тимчук (ХНТУСГ ім. П. Василенка).

Д 714 Доценко С. І. Людино-машинний інтерфейс: навч. посібник. – Харків: УкрДУЗТ, 2022. – 135 с., рис. 96, табл. 1.

ISBN

Викладено теоретичні основи та практичні рекомендації з проектування людино-машинних інтерфейсів.

Навчальний посібник призначено для здобувачів вищої освіти денної та заочної форм навчання для освітніх програм 1-го рівня, що навчаються за спеціальностями 123 «Комп'ютерна інженерія», 126 «Інформаційні системи і технології».

Навчальний посібник призначений для студентів закладів вищої освіти і фахівців у галузі інформаційних технологій.

УДК 004.65(075)

ISBN

ЗМІСТ

ВСТУП

Дисципліна «Людино-машинні інтерфейси» є дисципліною профільної теоретичної та практичної підготовки і належить до циклу спеціальних дисциплін, забезпечує базову підготовку студентів. Навчальний посібник призначено для студентів денної та заочної форм навчання для освітніх програм 1-го та 2-го рівнів, що навчаються за спеціальністю 123 «Комп'ютерна інженерія», 126 «Інформаційні системи і технології», 151 «Автоматизація та комп’ютерно-інтегровані технології».

Основною метою при вивченні дисципліни є засвоєння студентами основних парадигм проектування високоякісних інтерфейсів користувача та ознайомлення з теоретичною і практичною базою, що використовується при вирішенні задач побудови користувацького інтерфейсу.

У розділах навчального посібника наведено теоретичні відомості:

– щодо задач, розв'язуваних із застосуванням технологій побудови людино-машинного інтерфейсу;

– про стан розвитку сучасних принципів проектування інтерфейсів, про проблеми й напрямки розвитку цього розділу програмування;

– про основні методи й засоби автоматизації проектування, використовуваних у програмних засобах.

Матеріал розділів навчального посібника сформовано на основі навчальних матеріалів навчального посібника з дисципліни «Людино-машинний інтерфейс» доц. Г. А. Уткіної [1], а також опорного конспекту лекцій з дисципліни «Сучасні інструментальні засоби розробки користувацького інтерфейсу» Р. П. Шевчука [2]. Водночас, з навчальних матеріалів виділено смислові одиниці знань для відповідних об’єктів інтерфейсів. Після цього для вказаних об’єктів визначено їхні основні властивості. Цей підхід дає змогу підвищити рівень структурування знань і, як наслідок, підвищити рівень їх засвоювання.

РОЗДІЛ 1. Елементи інформаційних систем

1.1. Визначення інформаційних систем

Програмна система – це програмна продукція, яка являє собою сукупність програм і (або) підсистем, що мають загальне цільове призначення. Зв'язок між програмами і (або) підсистемами встановлює розробник, користувач або інші фахівці [1]. На рис. 1.1 наведено варіанти визначення поняття «інформаційна система».

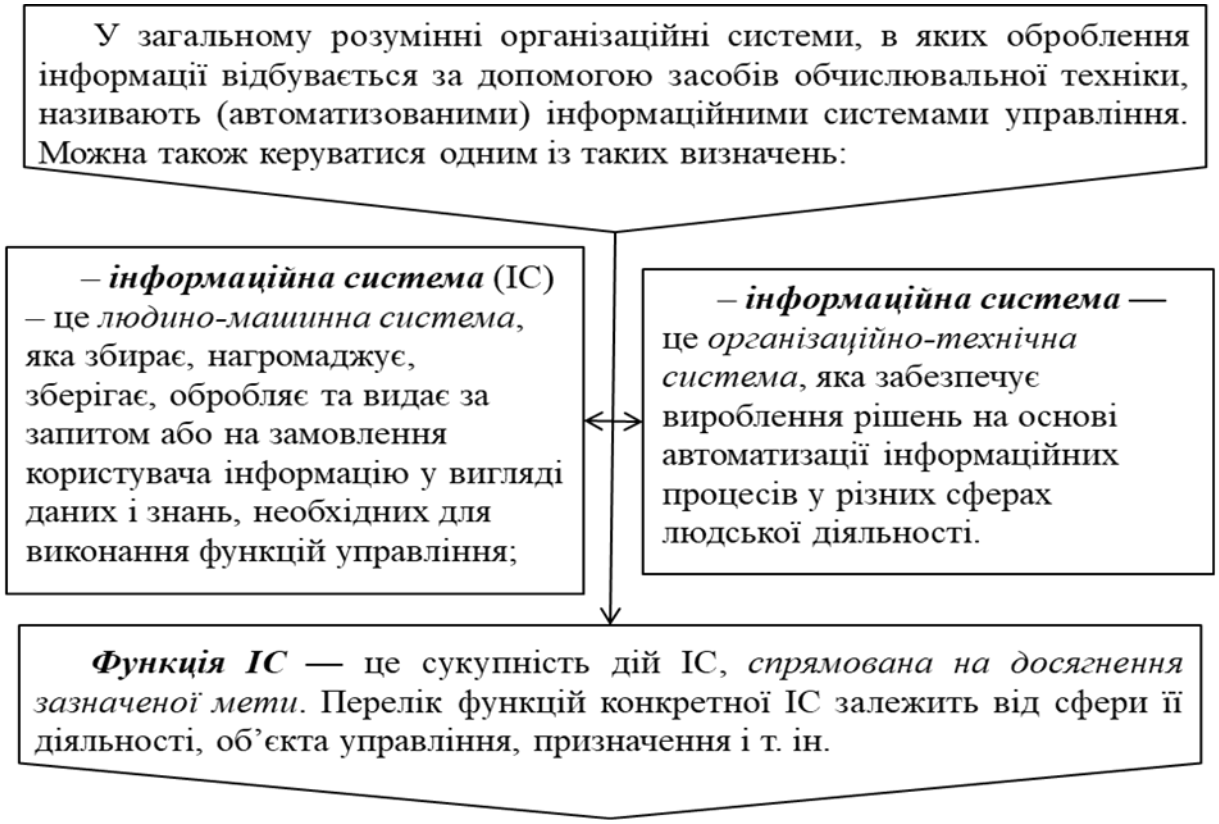

Рис. 1.1. Варіанти визначення поняття «інформаційна система»

Інформаційні системи насамперед поділяються за масштабом на одиночні, групові й корпоративні [1].

Одиночні інформаційні системи реалізуються на автономному комп'ютері, як правило, персональному. Така система може містити кілька простих додатків, зв'язаних загальним інформаційним фондом, і розрахована на роботу одного користувача чи групи користувачів, що розділяють за часом одне робоче місце.

Подібні додатки створюються за допомогою так званих настільних систем керування базами даних (FoxPro, Paradox, dBase, MS Access) або файлової системи і діалогової оболонки для введення, редагування й обробки даних.

Групові інформаційні системи орієнтовані на колективне використання інформації членами робочої групи (одного підрозділу), найчастіше будуються як локальна обчислювальна мережа ПК або (рідше), як багатотермінальна централізована обчислювальна система.

Однотипні чи спеціалізовані робочі місця забезпечують виклик одного чи декількох конкретних додатків.

Загальний інформаційний фонд являє собою базу даних чи сукупність файлів документів.

Спільне використання інформації організується за допомогою блокувань записів і файлів.

При розробці таких додатків використовуються настільні СУБД, сервери БД для робочих груп (Btrieve, NetWare SQL, Gupta SQLBase, Sybase Anywhere SQL, MS SQL Server, Progress, Informix-SE, Workgroup Oracle та ін.) і відповідні інструменти розробки чи системи керування документами та їхні інструментальні засоби.

Корпоративні інформаційні системи є розвитком систем для робочих груп і орієнтовані на масштаб компанії, можуть підтримувати територіально рознесені вузли чи мережі.

Вони можуть мати ієрархічну структуру з декількох рівнів.

Головна особливість – забезпечення доступу з підрозділу до центральної чи розподіленої бази даних компанії, крім доступу до інформаційного фонду робочої групи.

Для таких систем характерна архітектура *«клієнт – сервер»* зі *спеціалізацією серверів*.

Вони будуються на корпоративних SQL-серверах баз даних (Oracle7, Informix-OnLine, Informix-DSA, Sybase, CA-Ingress і ін.) і відповідних інструментальних засобах.

За оперативністю обробки даних розрізняють такі основні типи інформаційних систем – *пакетні* й *оперативні* інформаційні системи (реального часу) [1].

Інформаційні системи з пакетною обробкою в чистому вигляді можна зустріти на великих централізованих ЕОМ.

Інформаційним системам оперативної обробки трансакцій OLTP (Online Transaction Processing) властивий режим відображення актуального стану предметної сфери в будь-який момент часу, а пакетна обробка займає дуже обмежену нішу. Для систем OLTP характерний регулярний (можливо, інтенсивний) потік досить простих трансакцій, що виконують функцію замовлень, платежів, запитів тощо.

Важливими вимогами є висока продуктивність обробки трансакцій і гарантована доставка інформації при віддаленому доступі до баз даних по телекомунікаціях [1, 2]. Важливу роль відіграють інструменти інформаційних технологій. Їх склад визначено на рис. 1.2.

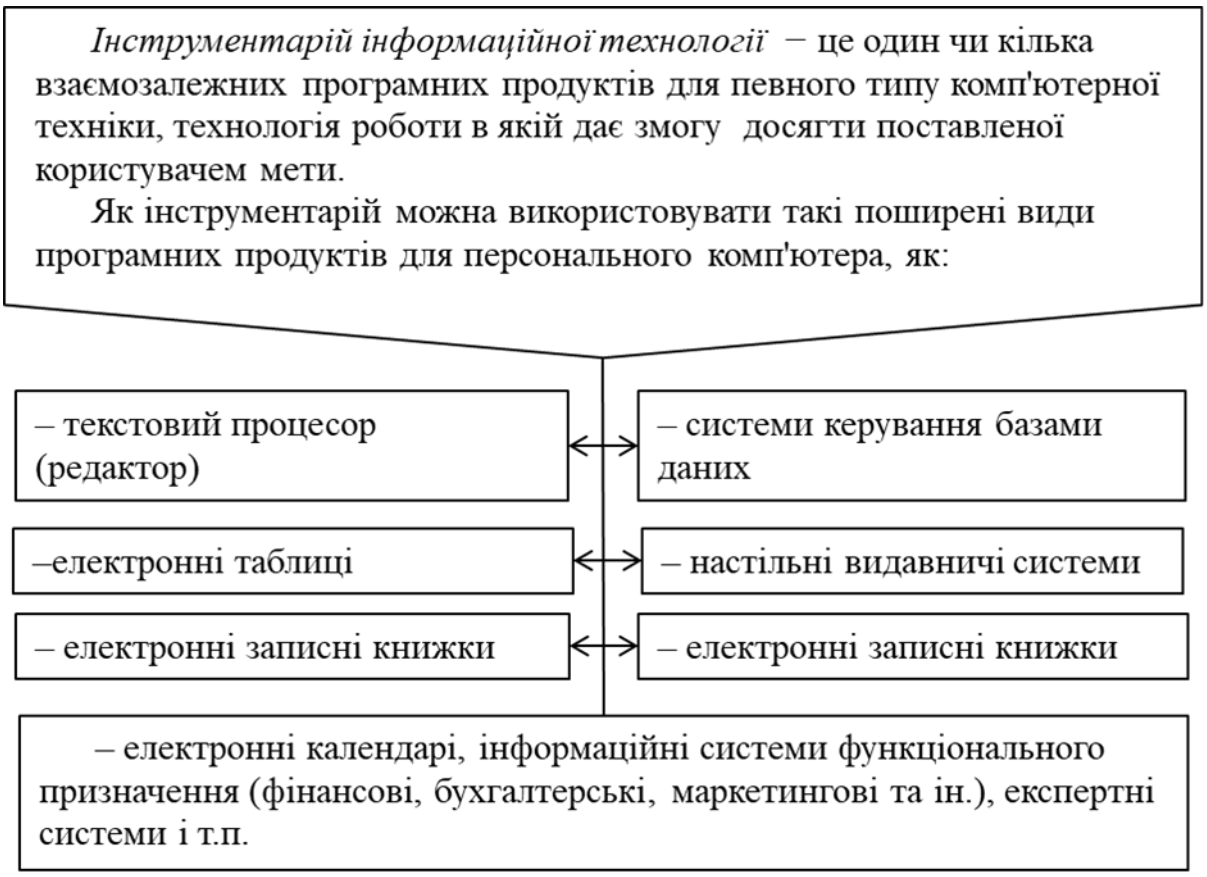

Рис. 1.2. Склад інструментів інформаційних технологій

Класифікація інформаційних технологій може стати *методологічною основою* їхнього вибору і використання при вирішенні завдань управлінської діяльності. Інформаційні технології класифікують за такими типами.

1. За функціями забезпечення управлінської діяльності:

- технології підготовки текстових документів на основі текстових процесорів;
- технології підготовки ілюстрацій і презентацій на основі графічних процесорів;
- технології підготовки табличних документів на основі використання табличних процесорів;
- технології розробки програм на основі алгоритмічних, об'єктно-орієнтованих і логічних мов програмування;
- технології систем керування базами даних;
- технології підтримки управлінських рішень на основі систем штучного інтелекту;
- гіпертекстові технології і технології мультимедіа.

2. За типом інтерфейсу користувача:

- ✓ командний;
- ✓ графічний інтерфейс користувача;
- ✓ інтерфейс пошукових систем.

3. За ступенем просторової взаємодії та за способом побудови мережі, що відбиває ту чи іншу форму і ступінь використання обчислювальних мереж:

- локальні;
- багаторівневі;
- розподілені.

4. За предметними галузями, що обслуговуються:

- бухгалтерський облік;
- банківська діяльність;
- податкова діяльність;
- страхова діяльність та ін.

1.2. Специфіка інформаційних систем

Важливою особливістю інформаційних систем є те, що вони володіють двома спільними для всіх інформаційних систем властивостями. Їх зміст розкрито на рис. 1.3.

Зауважимо, що обчислювальні програмні системи не обов'язково мають розвинені *інтерфейси*. Звичайно, це залежить від ступеня відчужуваного програмного продукту. Якщо система

призначена для продажу, то вона повинна мати хороший інтерфейс хоча б для маркетингу [1, 2].

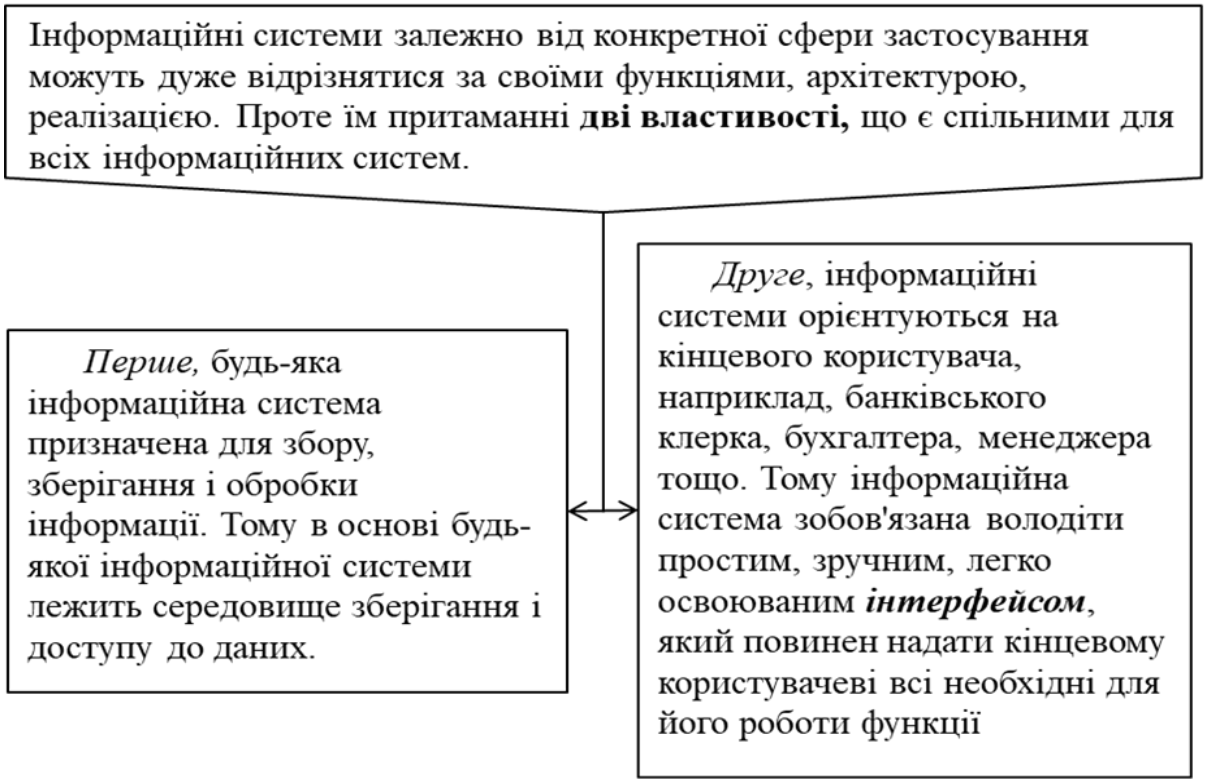

Рис. 1.3. Зміст спільних для всіх інформаційних систем властивостей

1.3. Основні задачі, які вирішують інформаційні системи

Конкретні задачі, які повинна вирішувати інформаційна система, залежать від тієї прикладної галузі, для якої призначена система. Сфери застосування інформаційних додатків різноманітні: банківська справа, страхування, медицина, транспорт, освіта тощо. Важко вказати галузь ділової активності, у якій сьогодні можна було б обійтися без використання інформаційних систем.

На практиці можна виділити деяку кількість задач, які *не залежать від специфіки прикладної галузі* (рис. 1.4). Природно, такі задачі пов'язані із загальними рисами інформаційних систем, розглянутих у попередньому пункті [1].

Рис. 1.4. Зміст задач, які не залежать від специфіки прикладної галузі

На перший погляд зазначена задача здається мало істотною. Вважається, що якщо інформаційна система забезпечує повний набір функцій і її *інтерфейс* забезпечує доступ до будь-якої з цих функцій, то кінцеві користувачі мають бути задоволені. Насправді це не так, бо користувачі часто судять про *якість системи в цілому*, зважаючи на якість *її інтерфейсу*.

1.4. Визначення змісту поняття «інтерфейс»

Комп'ютер – це машина, а не біологічна істота. Як будь-який технічний пристрій, комп'ютер обмінюється інформацією з людиною за допомогою набору певних правил, обов'язкових як для машини, так і для людини. Ці правила в комп'ютерній літературі називаються інтерфейсом [1].

Інтерфейс (від англ. interface — межа розділу, стик, область контакту або взаємодії) – ***це правила взаємодії операційної системи з користувачами****, а також сусідніми рівнями в мережі ЕОМ.*

Від інтерфейсу залежить технологія спілкування людини з комп'ютером.

Оскільки інтерфейс – це *набір правил*, а будь-які правила можна узагальнити, зібрати до «кодексу», згрупувати за загальною ознакою. Таким чином, ми прийшли до поняття «*вид інтерфейсу» як об'єднання за схожістю способів взаємодії людини і комп'ютера* [1].

Залежно від сутності взаємодії розрізняють різні види інтерфейсів. Розглянемо сучасні види інтерфейсів.

Командний інтерфейс. Командний інтерфейс називається так, бо в цьому виді інтерфейсу людина подає «команди» комп'ютеру, а комп'ютер їх виконує і видає результат.

WIMP – інтерфейс (Window – вікно, Image – образ, Menu – меню, Pointer – покажчик). Характерною особливістю цього виду інтерфейсу є те, що взаємодія з користувачем ведеться не за допомогою команд, а за допомогою *графічних образів* – меню, вікон інших елементів.

SILK – інтерфейс (Speech – мова, Image – образ, Language – мова, Knowlege – знання). Цей вид інтерфейсу найбільш наближений до звичайної, людської форми спілкування. У рамках цього інтерфейсу йде звичайна «розмова» людини і комп'ютера. При цьому комп'ютер знаходить для себе команди, аналізуючи людську мову і знаходячи в ній ключові фрази.

Закономірно, що кожен з видів інтерфейсів вимагає певної структури реалізації – технології.

> Технологія це - комплекс наукових та інженерних знань, реалізованих у прийомах праці, наборах матеріальних, технічних, енергетичних, трудових факторів виробництва, засобах їх об'єднання для створення продукту або послуги, що відповідають певним вимогам.

Розглянемо основні аспекти реалізації технології відповідно до виду інтерфейсу [1].

Командний інтерфейс реалізований у вигляді *пакетної технології* і *технології командного рядка*.

Пакетна технологія. Історично цей вид технології з'явився першим. Вона існувала вже на релейних машинах Зюса і Цюзе (Німеччина, 1937 рік).

Ідея її проста: на вхід комп'ютера подається послідовність символів, у яких за певними правилами задається послідовність запущених на виконання програм.

Після виконання чергової програми запускається наступна і так далі. Машина за певними правилами знаходить для себе команди й дані.

В образі послідовності може виступати, наприклад, перфострічка, набір перфокарт, послідовність натиснення клавіш електричної друкарської машинки (типу CONSUL).

Машина також видає свої повідомлення на перфоратор, алфавітно-цифровий друкувальний пристрій (АЦДП), стрічку друкарської машинки.

Технологія командного рядка. При цій технології як єдиний спосіб введення інформації від людини до комп'ютера є *клавіатура*, а комп'ютер виводить інформацію людині за допомогою *алфавітно-цифрового дисплея (монітора).*

Цю комбінацію (монітор + клавіатура) називають терміналом, або консоллю.

Команди набираються в командному рядку. Командний рядок є символом запрошення та прямокутником зі змінним освітленням.

Команда закінчується натисненням клавіші Enter (або Return). Після цього здійснюється перехід на початок наступного рядка.

Саме з цієї позиції комп'ютер видає на монітор результати своєї роботи. Потім процес повторюється.

Графічний інтерфейс. Ідея графічного інтерфейсу з’явилася в середині 70-х років XX століття, коли в дослідницькому центрі Xerox Palo Alto Research Center (PARC) була розроблена концепція *візуального інтерфейсу*.

Передумовою графічного інтерфейсу було зменшення часу реакції комп'ютера на команду, збільшення обсягу оперативної пам'яті, а також розвиток технічної бази комп'ютерів.

Апаратною основою концепції, звичайно ж, була поява алфавітно-цифрових дисплеїв на комп'ютерах, причому на цих дисплеях уже були такі ефекти, як «мерехтіння» символів, інверсія кольору, підкреслення символів.

Наступним кроком у розвитку графічного інтерфейсу стало створення кольорового дисплея. Це дало змогу виводити, разом з цими ефектами, символи в 16 кольорах на фоні з палітрою з 8 кольорів.

Спочатку візуальний інтерфейс використовувався тільки в програмах. Поступово він почав переходити і на операційні

системи. З часом проходив процес з уніфікації у використанні клавіатури і миші прикладними програмами.

Злиття цих двох тенденцій і привело до створення того призначеного для користувача інтерфейсу, за допомогою якого, при мінімальних витратах часу і засобів на перенавчання персоналу, можна працювати з будь-яким програмним продуктом.

Графічний інтерфейс користувача за час свого розвитку пройшов два етапи [1].

На першому етапі розвитку графічний інтерфейс набув таких особливостей:

- *виділення областей екрана;*
- *перевизначення клавіш клавіатури залежно від контексту;*
- *використання маніпуляторів і «сірих» клавіш клавіатури для управління курсором;*
- *широке використання кольорових моніторів.*

Поява цього типу інтерфейсу збігається з великим поширенням операційної системи MSDOS. Саме вона втілила цей інтерфейс у масову практику, завдяки чому 80–ті роки пройшли під знаком удосконалення цього типу інтерфейсу, поліпшення характеристик відображення символів та інших параметрів монітора.

Другим етапом у розвитку графічного інтерфейсу став «чистий» інтерфейс WIMP. Цей підвид інтерфейсу характеризується такими особливостями:

– уся робота з програмами, файлами і документами відбувається у вікнах певних обкреслених рамкою частинах екрана;

– усі програми, файли, документи, пристрої й інші об'єкти подаються у вигляді значків – ікон. При відкритті ікони перетворюються на вікна;

– усі дії з об'єктами здійснюються за допомогою меню. У чистому WIMP – інтерфейсі меню стає основним елементом управління;

– широке використання маніпуляторів для вказівки на об'єкти.

Маніпулятор перестає бути просто іграшкою – доповненням до клавіатури, а стає основним елементом управління.

За допомогою маніпулятора вказують на будь-яку область екрана, вікна або ікони, виділяють її, а вже потім через меню або з використанням інших технологій, здійснюють управління ними.

Яскравим прикладом програм з графічним інтерфейсом є операційна система Microsoft Windows.

Із середини 90-х років XX століття після появи недорогих звукових карт і великого поширення технологій розпізнавання мови з'явилася так звана *«мовна технологія» SILK – інтерфейсу*. За цією технологією команди подаються голосом шляхом вимовляння спеціальних зарезервованих слів – команд. Основними такими командами (за правилами системи «Горинич») є [1]:

– «Прокинься» – увімкнення голосового інтерфейсу;

– «Відпочивай» – вимкнення мовного інтерфейсу;

– «Відкрити» – перехід у режим виклику тієї або іншої програми. Ім'я програми називається в наступному слові;

– «Диктуватиму» – перехід з режиму команд у режим набору тексту голосом;

– «Режим команд» – повернення в режим подачі команд голосом.

Слова повинні вимовлятися чітко, в одному темпі. Між словами обов'язкова пауза. Через недосконаліть алгоритму розпізнавання мови такі системи вимагають індивідуального попереднього настроювання на кожного конкретного користувача.

«Мовна» технологія є простою реалізацією SILK – інтерфейсу [1].

Біометрична технологія («Мімічний інтерфейс».) Ця технологія виникла в кінці 90–х років XX століття.

Для *управління комп'ютером* використовується *вираз обличчя* людини, *напрям його погляду, розмір зіниці і інші ознаки*.

Для *ідентифікації користувача* використовується *малюнок райдужної оболонки його очей*, *відбитки пальців* і інша унікальна інформація.

Зображення прочитуються з цифрової відеокамери, а потім за допомогою спеціальних *програм розпізнавання образів* з цього зображення виділяються команди.

Ця технологія, мабуть, займе своє місце в програмних продуктах і додатках, де важливо точно ідентифікувати користувача комп'ютера.

Семантичний (Суспільний) інтерфейс. Цей вид інтерфейсу виник у кінці 70-х років XX століття з розвитком штучного інтелекту. Його важко назвати самостійним видом інтерфейсу: він включає і інтерфейс командного рядка, і графічний, і мовний, і мімічний інтерфейс. Основна його відмінність – це *відсутність команд при спілкуванні з комп'ютером*. Запит формується природною мовою, у вигляді зв'язаного тексту та образів.

По своїй суті це важко називати інтерфейсом – це вже *моделювання «спілкування» людини зкомп'ютером* [1].

Крім того, вибір типу інтерфейсу включає вибір технології роботи з документами. Розрізняють дві технології:

– однодокументна, яка припускає однодокументний інтерфейс (SDI - Single Document Interface);

– багатодокументна, яка припускає багатодокументний інтерфейс (MDI - Multiple Document Interface).

1.5. Характеристики процесу проектування користувацького інтерфейсу

На загальному рівні проектування КІ здається простою справою. Дійсно, спроектувати будь-який фрагмент КІ не складно, однак мало хто може впоратись з розробкою повного інтерфейсу з урахуванням усіх аспектів аж до етапу розгортки КІ [1]. Склад характеристик процесу проектування користувацького інтерфейсу наведено на рис. 1.5.

Рис. 1.5. Характеристика процесу проектування користувацького інтерфейсу

Складність − звичайна властивість ПЗ, але в ще більшому ступені це стосується КІ − через численні чинники та невизначеності, які впливають на його розробку.

Проектування КІ − *нелінійний процес*, оскільки існування фіксованого, впорядкованого і прямолінійного шляху від початку до кінця зовсім не обов'язкове.

Процес проектування відрізняється *невизначеністю*, оскільки не існує рівняння, за яким можна було б одержати однаковий результат при заданих однакових початкових умовах, крім того, практично неможливо одержати ідентичний результат навіть під примусом.

Користувацький інтерфейс *неортогональний* у тому сенсі, що будь-який аспект проектного розв'язку може впливати на інші аспекти, причому результат цього впливу не завжди є приємним сюрпризом.

Важливою особливістю процесу проектування користувацького інтерфейсу є його спрямованість на користувача.

Програмний КІ привертає все більшу увагу і набуває все більшого значення як складова *конкурентної переваги*.

Існує декілька умов, які дають змогу говорити про те, що проект ведеться в орієнтованому на *користувача стилі*:

– *розуміння користувачів та їх задач, залучення користувачів* в усі аспекти життєвого циклу продукту;

– *постановка цілей, які можна виміряти*; встановлення критеріїв успіху з точки зору користувачів та підприємств;

– проект повинен передбачати *нову компетентність користувача*, яка стосовно продукту включає пакетування, маркетинг, навчання, віддруковану інформацію, налагодження

параметрів, інсталяцію, екрани, графіку, довідки, іншу експлуатаційну підтримку, оновлення та деінсталяцію;

– *оцінювання та тестування* за участю *реальних користувачів* для визначення, чи досягнуті цілі та які проблеми існують;

– *ітеративний підхід* − якщо цілі не досягнуті або існують проблеми, слід внести виправлення та провести повторну перевірку.

Важливою обставиною при розробці користувацького інтерфейсу є обрання ефективного підходу до його проектування. На рис. 1.6 наведено склад та зміст основних підходів до проектування.

Рис. 1.6. Склад та зміст основних підходів до проектування

Найкращий підхід до розробки – *еволюційний ітеративний підхід «ззовні всередину»* [1].

Орієнтоване на користувача проектування продукту – міждисциплінарний та ітеративний процес розробки ПЗ, спрямований на досягнення користувацьких цілей стосовно продукту, на практичність та інші вимірювані властивості продукту протягом його життєвого циклу [1].

На рис. 1.7 наведено склад та зміст основних етапів проектування користувацького інтерфейсу, а також визначена послідовність реалізації визначених етапів.

Рисунок 1.7. Склад та зміст основних етапів проектування користувацького інтерфейсу

Важливою інтегральною характеристикою користувацького інтерфейсу є задоволеність користувача.

На рис. 1.8 наведено склад та зміст чинників, які формують та визначають зміст поняття «задоволеність користувача».

Користувач або представник компанії не завжди може чітко визначити свої потреби й вимоги на етапі планування та аналізу вимог, однак він може бути дуже категоричним відносно того, що

бажано й небажано в програмному продукті, коли він уже стане доступним для апробації й використання [1, 2].

Рис. 1.8. Склад та зміст чинників, які формують та визначають зміст поняття «задоволеність користувача»

Щодо термінів розробки, то це питання кваліфікації та досвіду стосовно сучасної технології КІ. З множиною стилів та функцій КІ пов'язаний великий обсяг інформації, який включає масу деталей, особливості очікуваної поведінки, а також надлишкові відомості.

1.6. Форми стилів користувацького інтерфейсу

Існує ряд стилів КІ, які завоювали популярність в індустрії програмних засобів [1]. На рис. 1.9 наведено характеристику стилю графічного користувацького інтерфейсу, а на рис. 1.10 наведеного характеристику користувацького WEB інтерфейсу.

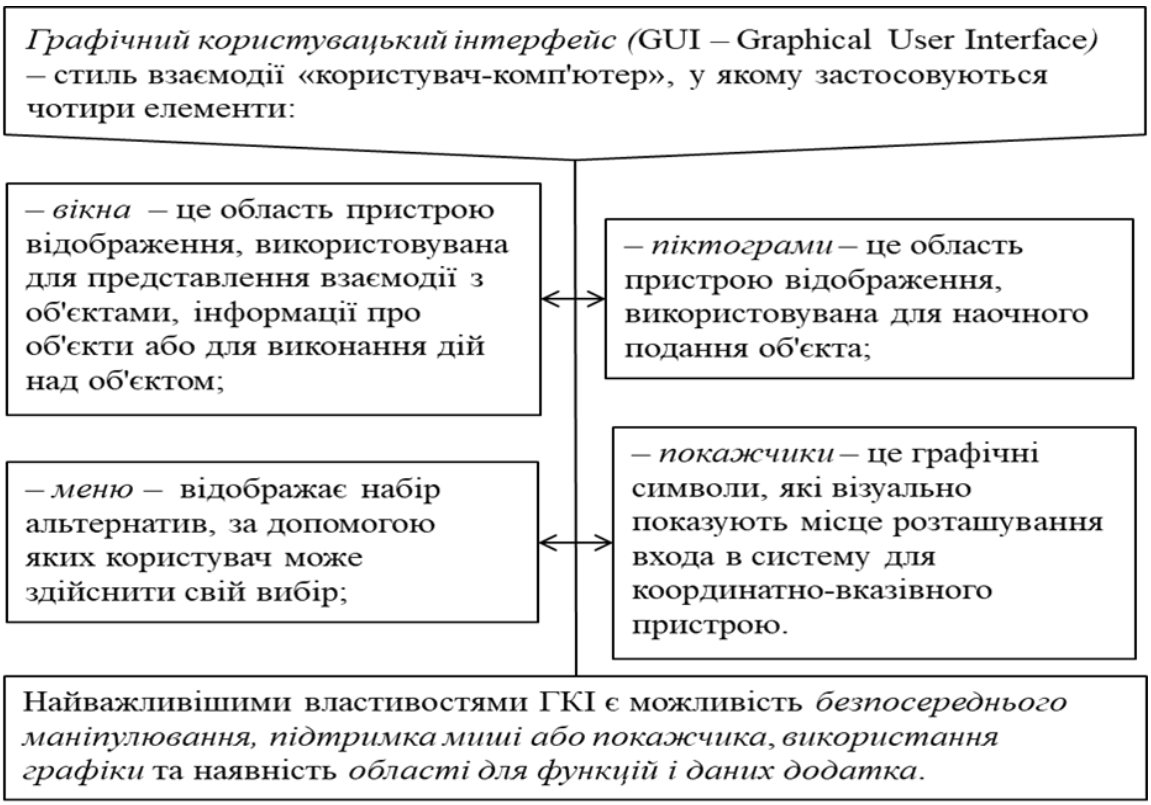

Рис. 1.9. Характеристика графічного користувацького інтерфейсу

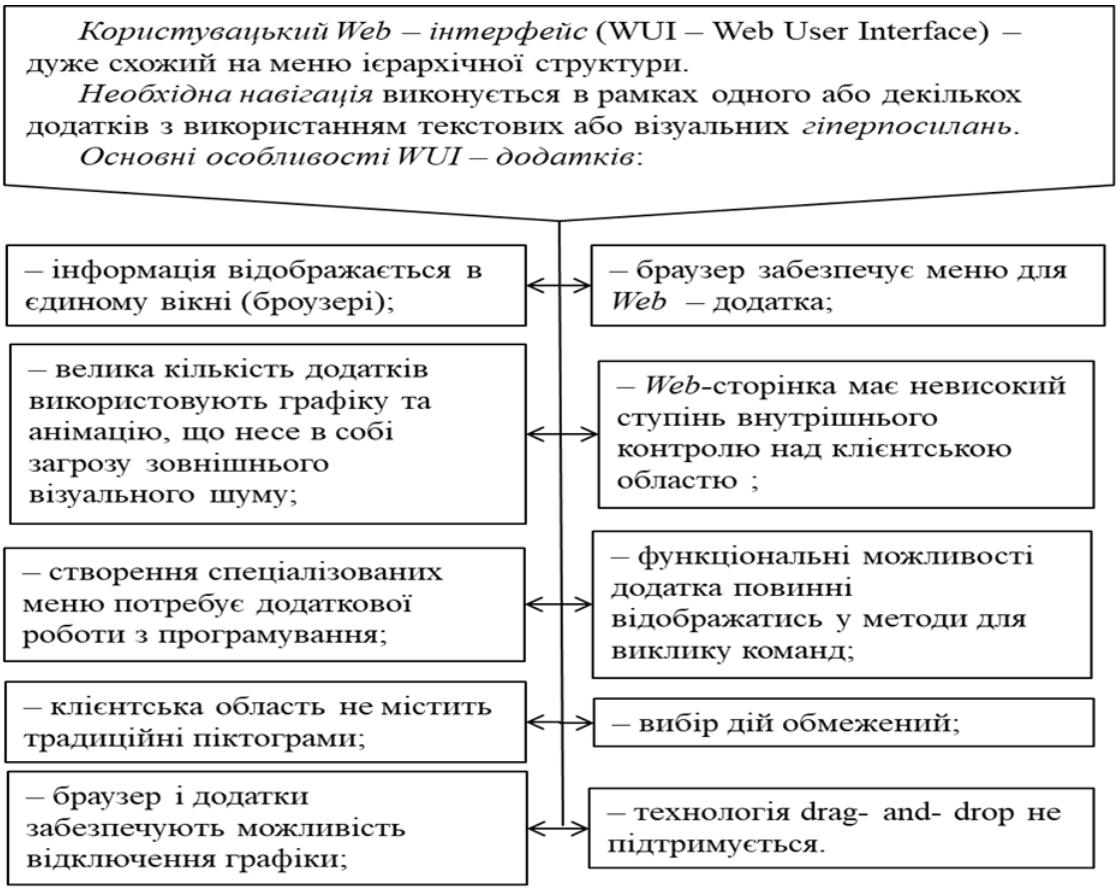

Рис. 1.10. Характеристика стилю WEB інтерфейсу

Склад елементів користувацького WEB інтерфейсу наведено на рис. 1.11.

Рис. 1.11. Склад елементів користувацького WEB інтерфейсу

Користувацький інтерфейс кишенькових пристроїв (HUI) – користувацький інтерфейс кишенькових комп'ютерів, для введення даних у яких застосовують «жестикуляційний» стиль з пером та сенсорним маленьким екраном. HUI – інтерфейс забезпечує деякі можливості ГКІ, а саме: піктограми меню та аналогічна поведінка покажчика [1].

Прикладний рівень КІ програмного забезпечення – GUI, WUI, HUI –інтерфейси являють собою загальні стилі КІ. До чинників, пов'язаних з прикладним КІ, які впливають на практичність, належить спосіб використання GUI – стилю з компонентами КІ прикладного рівня. Побудований на основі КІ додаток тією чи іншою мірою використовує компоненти стилю КІ або компоненти КІ прикладного рівня [1].

На рис. 1.12 наведено можливості прикладного рівня користувацького інтерфейсу програмного забезпечення.

Рис. 1.12. Можливості прикладного рівня користувацького інтерфейсу програмного забезпечення

Важливим видом користувацького інтерфейсу є об’єктно-орієнтовані користувацькі інтерфейси. Їх основні властивості вказано на рис. 1.13.

Рис.1.13. Основні властивості об’єктно-орієнтованих користувацьких інтерфейсів

1.7. Моделювання користувацького інтерфейсу

Не може бути одного найкращого інструменту, найкращої програми, інтерфейсу для користувача комп'ютера, оскільки цілі користувача постійно змінюються залежно від поставлених у цей момент задач. Користувачі потребують не лише різних інтерфейсів для вирішення різних задач, а й зміни функціональності програми навіть під час виконання однієї задачі. *Немає такої програми чи інтерфейсу, які б відповідали всім потребам користувача в усі часи* [1].

Розглянемо чотири моделі КІ: ментальну, користувацьку, програміста, проектувальника [1].

Ментальна, або концептуальна модель – лише внутрішнє відображення того, як користувач розуміє і взаємодіє із системою (рис. 1.14). Це відображення фізичної системи або ПЗ, у якому закладено ймовірну послідовність дій при виконанні операцій введення та виведення.

Ментальна модель необов'язково точно відображає ситуацію та її компоненти, вона допомагає передбачати, що відбудеться далі.

Рис. 1.14. Основні властивості ментальних моделей користувацького інтерфейсу

Особливості моделі користувача користувацького інтерфейсу наведено на рис. 1.15. Спілкуватись потрібно зі справжніми користувачами, а не з їхніми менеджерами та керівництвом.

Рис.1.15. Особливості моделі користувача

Серед чотирьох визначених моделей модель програміста є найпростішою.

Модель програміста – найлегша для відображення, оскільки вона може бути формально і недвозначно описана. Ця модель є функціональною специфікацією програмного продукту.

Модель проектувальника. Проектувальник дізнається про ідеї, побажання користувача, поєднує їх зі своїми навичками та матеріалами, необхідними для програміста, та проектує ПЗ, яке повинно задовольняти потреби користувача. Модель проектувальника – це дещо середнє між моделлю користувача та моделлю програміста [1].

Розробники, як правило, не входять у контакт з користувачами, для яких створюють програми. *З'єднувачем* між користувацьким оточенням та програмістським світом є *проектувальник КІ*.

Модель проектувальника описує об'єкти, з якими працює користувач, і техніку маніпулювання ними.

Контрольні запитання до розділу 1

1. Наведіть визначення інформаційної системи як виду програмної системи.
2. Наведіть специфіку інформаційних систем.
3. Наведіть основні завдання інформаційних систем.
4. Наведіть визначення поняття «інтерфейс». .
5. Вкажіть особливості проектування користувацького інтерфейсу.
6. Наведіть стилі користувацького інтерфейсу.
7. Наведіть визначення моделі користувацького інтерфейсу.
8. Наведіть види інтерфейсів.

РОЗДІЛ 2. Моделі користувацьких інтерфейсів

2.1. Моделювання взаємодії комп'ютера і користувача

Під діалогом у цьому випадку розуміють регламентований обмін інформацією між людиною і комп'ютером, здійснюваний у реальному масштабі часу і спрямований на сумісне розв'язання конкретної задачі: обмін інформацією і координація дій [1].

Кожен діалог складається з окремих процесів введення-виведення, які фізично забезпечують зв'язок користувача і комп'ютера. На рис. 2.1 наведено класифікацію форм повідомлень, які складають зміст процесів обміну інформацією.

Рис. 2.1. Класифікація повідомлень, які складають зміст процесів обміну інформацією

Схематично організацію взаємодії користувача і машини подано на рис. 2.2 [1].

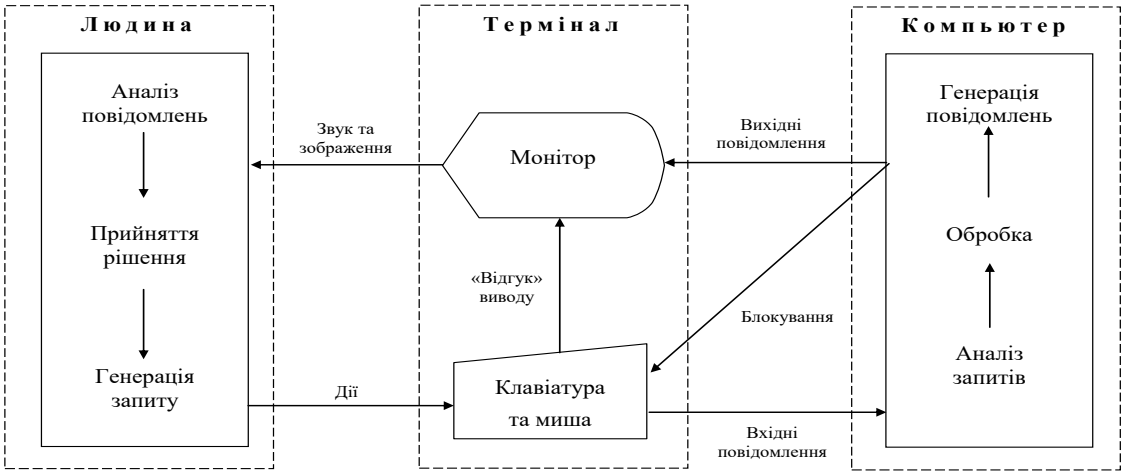

Рис. 2.2. Організація взаємодії комп'ютера і користувача

На рис. 2.3 наведено класифікацію типів повідомлень, які складають зміст процесів обміну інформацією.

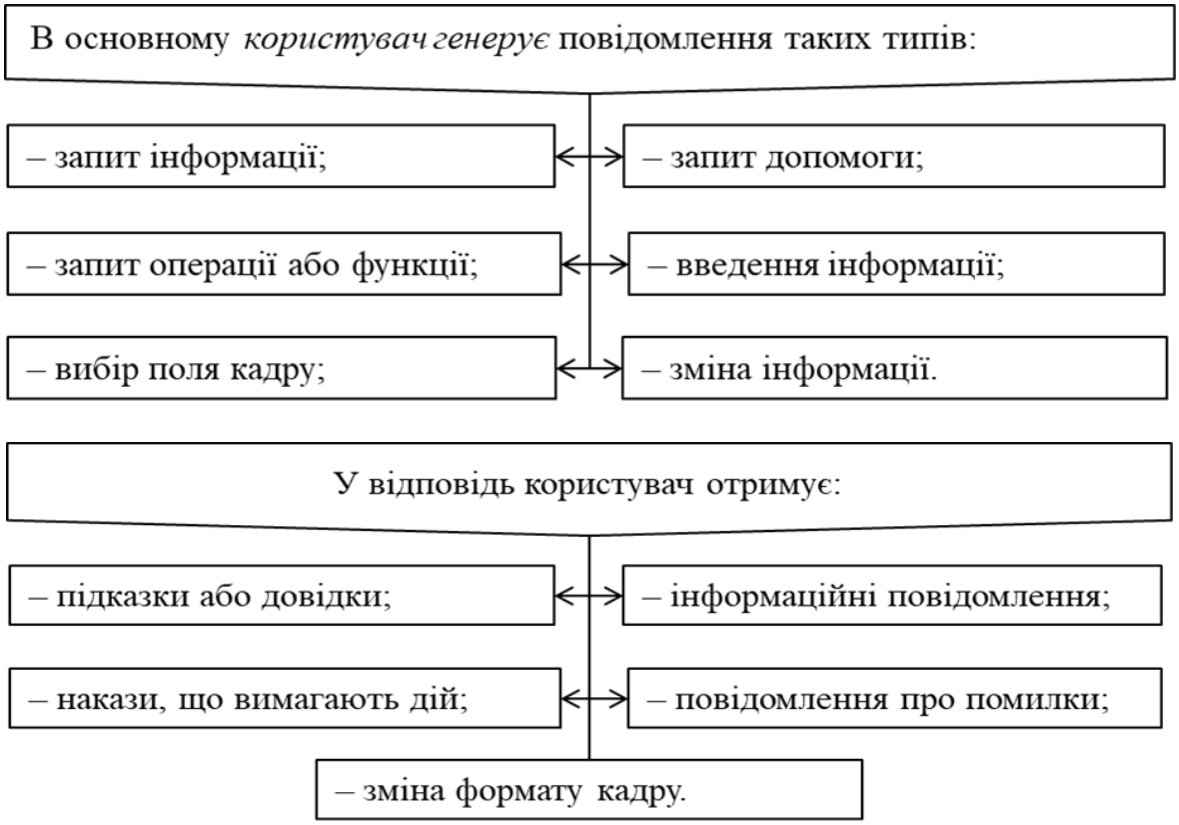

Рис. 2.3. Класифікація типів повідомлень

На рис. 2.4 наведено типи пристроїв, що забезпечують виведення повідомлень.

Рис. 2.4. Типи пристроїв, що забезпечують виведення повідомлень

На рис. 2.5 наведено типи пристроїв, що забезпечують введення повідомлень.

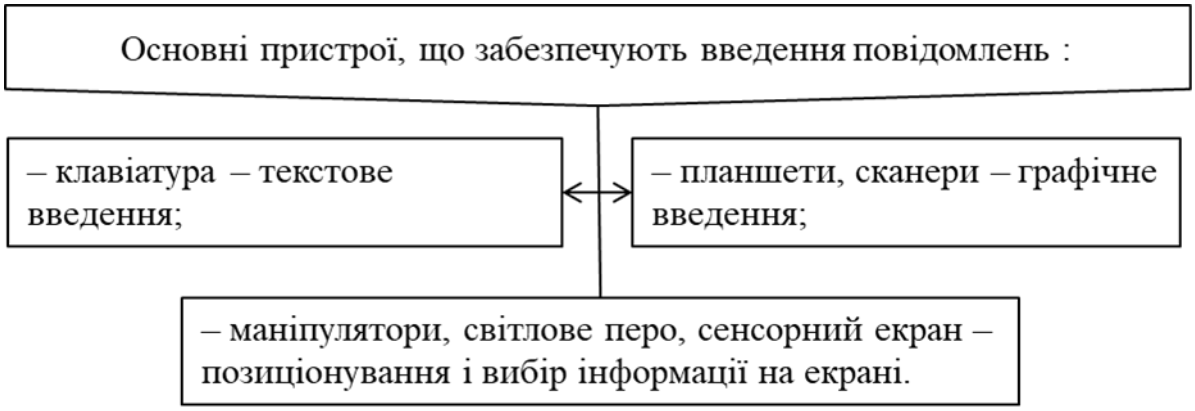

Рис. 2.5. Типи пристроїв, що забезпечують введення повідомлень

2.2. Класифікація користувацьких інтерфейсів за типом

За аналогією з процедурним і об'єктним підходом до програмування розрізняють процедурно-орієнтований і об'єктно-орієнтований підходи до розробки інтерфейсів – типи інтерфейсів [1].

Процедурно-орієнтовані інтерфейси використовують традиційну модель взаємодії з користувачем, основану на поняттях «*процедура*» й «*операція*».

У рамках цієї моделі програмне забезпечення надає користувачеві можливість виконання деяких дій, для яких користувач визначає відповідні дані і наслідком виконання яких є отримання бажаних результатів.

Об'єктно-орієнтовані інтерфейси використовують дещо іншу модель взаємодії з користувачем, орієнтовану на маніпулювання об'єктами наочної області.

У рамках цієї моделі *користувачеві надається можливість безпосередньо взаємодіяти з кожним об'єктом* та ініціювати виконання операцій, у процесі яких взаємодіють декілька об'єктів.

Завдання користувача формулюється як цілеспрямована зміна деякого об'єкта, що має внутрішню структуру, певний зміст і зовнішнє символьне або графічне подання.

Об'єкт при цьому розуміється в широкому сенсі слова, наприклад, *модель реальної системи або процесу, база даних, текст тощо*.

Користувачеві надається можливість створювати об'єкти, змінювати їх параметри і зв'язки з іншими об'єктами, а також ініціювати взаємодію цих об'єктів.

Елементи інтерфейсів цього типу включені в призначений для користувача інтерфейс Windows.

Наприклад, користувач може «узяти» файл і «перемістити» його в іншу теку. Таким чином він ініціює виконання операції переміщення файлу.

Порівняння вищезазначених типів інтерфейсів подано у табл. 2.1 [1].

Таблиця 2.1

Порівняння основних типів інтерфейсів

Процедурно-орієнтовані призначені для користувача інтерфейси	*Об'єктно-орієнтовані* призначені для користувача інтерфейси
Забезпечують користувачів функціями, необхідними для виконання завдань	Забезпечують користувачам можливість взаємодії з об'єктами
Акцент робиться на завдання.	Акцент робиться на вхідні дані і результати
Піктограми представляють додатки, вікна або операції	Піктограми представляють об'єкти
Зміст тек і довідників відображається за допомогою таблиць і списків	Теки і довідники є візуальними контейнерами об'єктів

Розрізняють *процедурно-орієнтовані* інтерфейси трьох підтипів: *примітивні*, з *меню*, з *вільною навігацією* [1].

Примітивним називають інтерфейс, який організовує взаємодію з користувачем у консольному режимі (рис. 2.6, а) [1].

Зазвичай такий інтерфейс реалізує конкретний сценарій роботи програмного забезпечення. Наприклад: введення даних – рішення задачі – виведення результату.

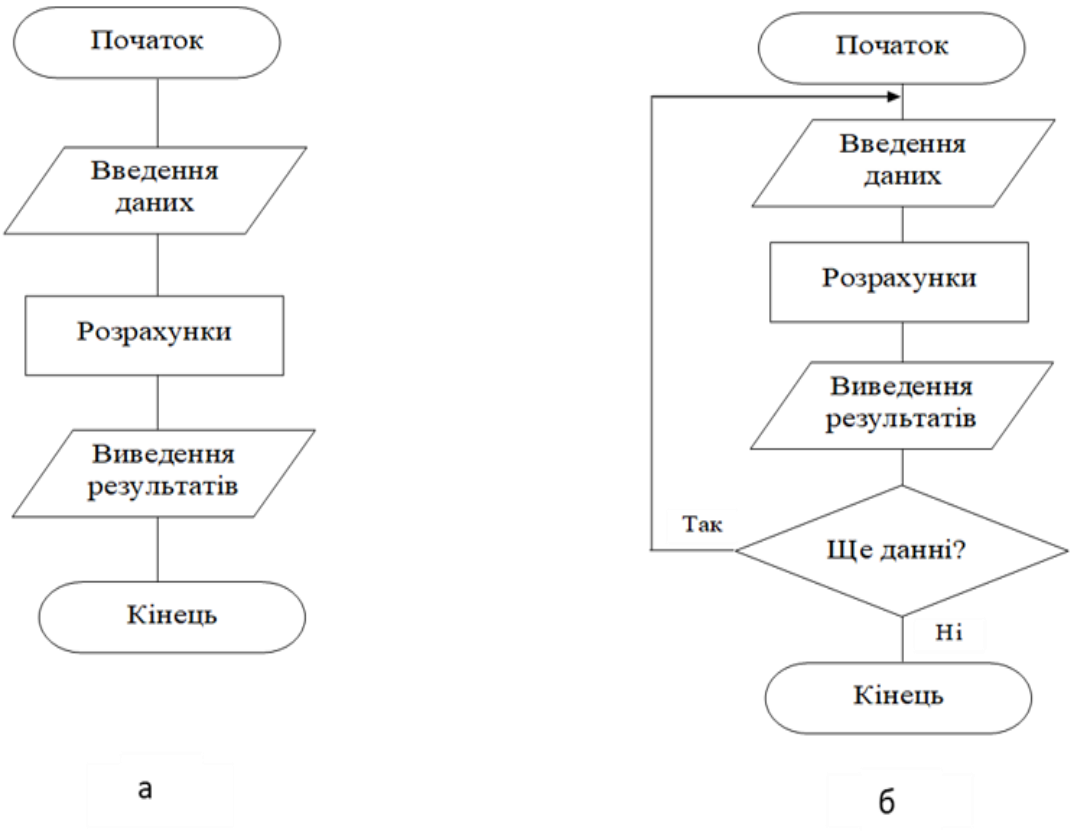

Рис. 2.6. Типова структура алгоритму програм з примітивним інтерфейсом: *а* – послідовний; *б* – з можливістю повторення

Подібні інтерфейси використовують тільки в процесі навчання програмуванню або в тих випадках, коли вся програма реалізує одну функцію, наприклад у деяких системних утилітах.

Інтерфейс-меню, на відміну від примітивного інтерфейсу, дає змогу користувачеві вибирати необхідні операції зі спеціального списку, який виводить йому програма. Ці інтерфейси передбачають реалізацію багатьох сценаріїв роботи, послідовність дій у яких визначає користувач.

Розрізняють *однорівневі* (рис. 2.7) та *ієрархічні* меню [1].

Перші використовують для порівняно простого *управління обчислювальним процесом*, коли варіантів небагато (не більше 5–7), і вони включають операції одного типу, наприклад, Створити, Відкрити, Закрити тощо. На рис. 2.7 наведено характеристики інтерфейсів користувача, об’єднаних схожістю розробки.

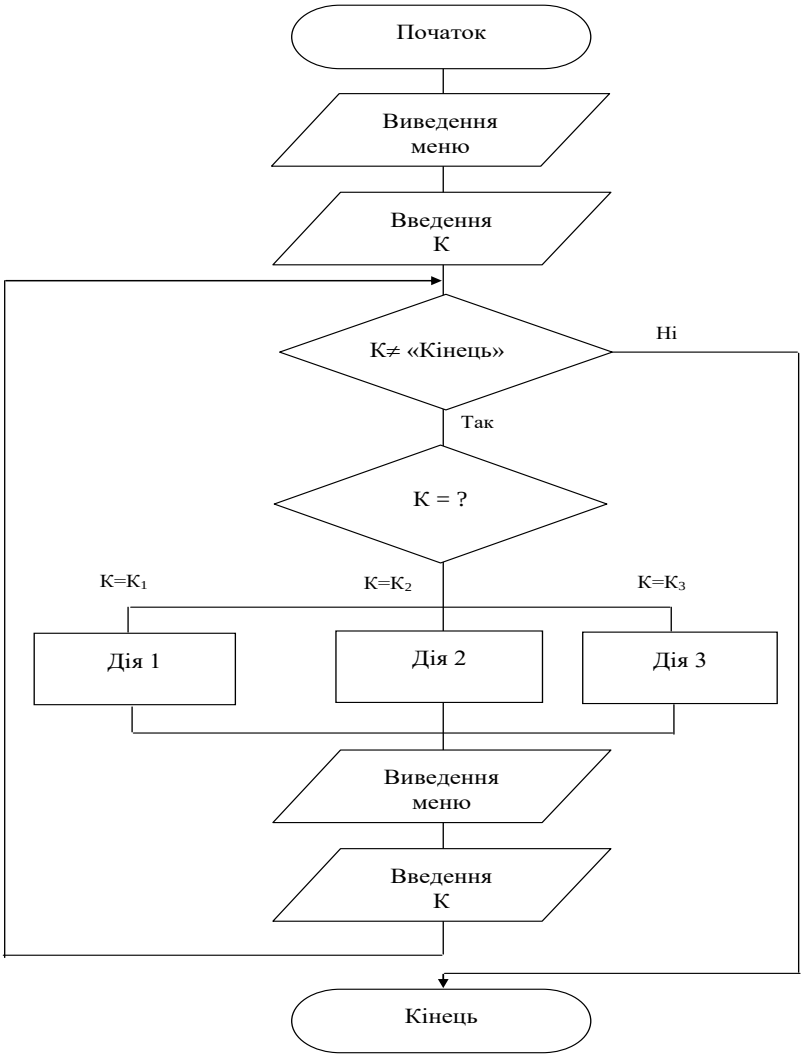

Рис. 2.7. Типова структура алгоритму програми з однорівневим меню

Ієрархічні – при великій кількості варіантів або їх очевидних відмінностях, наприклад, операції з файлами й операції з даними, що зберігаються в цих файлах.

Інтерфейс-меню передбачає, що програма перебуває або в стані *Рівень меню*, або в стані *Виконання операції*.

У стані *Рівень меню* здійснюється виведення меню відповідного рівня і вибір потрібного пункту меню, а в стані *Виконання операції* реалізується сценарій вибраної операції.

Як виняток іноді користувачеві надається можливість завершення операції незалежно від стадії виконання сценарію і/або програми, наприклад, за допомогою натиснення клавіші Esc.

Інтерфейси з вільною навігацією також називають *графічними* призначеними для користувача інтерфейсами (GUI –

Graphic User Interface) або інтерфейсами WYSIWYG (What You See Is What You Get – що бачиш, те й отримаєш, тобто що користувач бачить на екрані, те він і отримає при друку).

Ці назви підкреслюють, що інтерфейси цього типу орієнтовані на використання екрана в графічному режимі з високою роздільною здатністю.

Крім того, інтерфейси цього підтипу підтримують концепцію *сумісності програм*, даючи змогу переміщати між ними інформацію (технологія OLE).

Об'єктно-орієнтовані інтерфейси поки що представлені тільки *інтерфейсом прямого маніпулювання*.

Цей тип інтерфейсу передбачає, що взаємодія користувача з програмним забезпеченням *здійснюється за допомогою вибору і переміщення піктограм*, відповідних об'єктам наочної області.

Для реалізації таких інтерфейсів також використовують подієве програмування й об'єктно-орієнтовані бібліотеки.

Структурна схема інтерфейсів користувача, об’єднаних схожістю розробки, подана на рис. 2.8 [1].

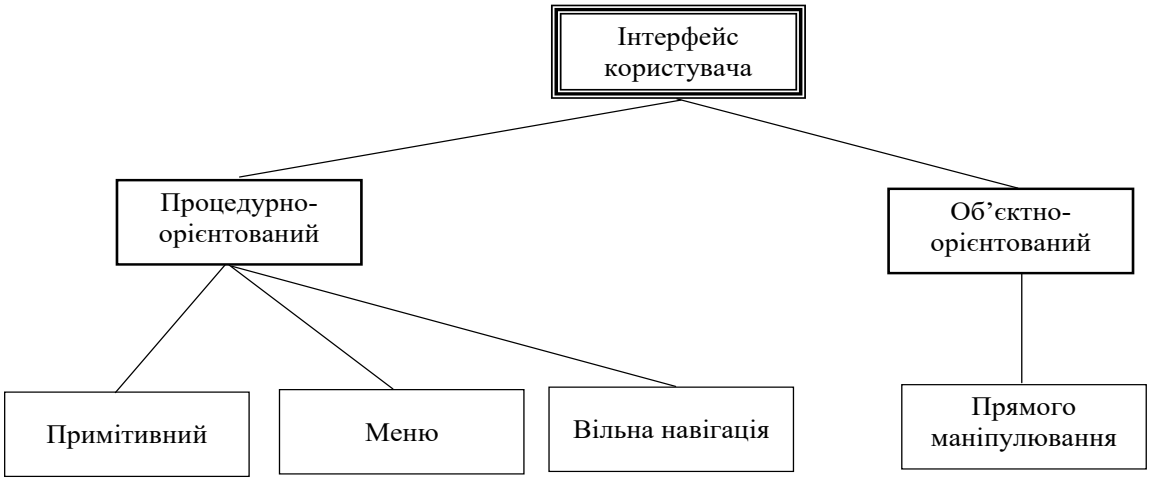

Рис. 2.8. Типи інтерфейсів, об’єднаних схожістю розробки

2.3. Приклади моделей інтерфейсів

Порівняємо реалізацію чотирьох указаних типів інтерфейсів на конкретному прикладі [1].

Приклад. Розробити користувацький інтерфейс згідно з технічним завданням для програмного продукту, призначеного

для наочної демонстрації графіків функцій однієї змінної $y = f(x)$. Програма повинна розраховувати таблицю значень і будувати графік функцій на заданому відрізку за заданою формулою і змінювати крок аргументу в межах відрізка. Окрім цього, програма повинна запам'ятовувати введені формули.

Побудуємо **чотири варіанти інтерфейсу**, відповідні розглянутим вище типам [1].

Варіант 1. Використання примітивного інтерфейсу припускає, що користувач відразу визначає всі параметри, необхідні програмі для побудови графіка або виведення таблиці, вводячи їх у відповідь на відповідні запити програми. Після чого програма виконує необхідні обчислення і виводить результат. Якщо допустити, що програма запрошуватиме підтвердження завершення обробки, то процес побудови графіків/таблиць можна зациклити. Залежно від використовуваних засобів отримаємо порівняно просту програму, що задовольняє функціональні специфікації, але орієнтовану на єдиний сценарій: введення – обробка – виведення (див. рис. 2.6, б). Цей варіант незручний для користувача, бо не виключена можливість зациклюваності процесу.

Варіант 2. Використання однорівневого меню, яке включатиме команди: Функція, Відрізок, Крок, Тип результату, Виконати і Вихід. При виборі першого пункту меню визначається функція, другого – інтервал, третього – крок, четвертого – тип результату, п'ятого – здійснюється операція, і, нарешті, останній пункт забезпечує можливість виходу з програми (рис. 2.9) [1].

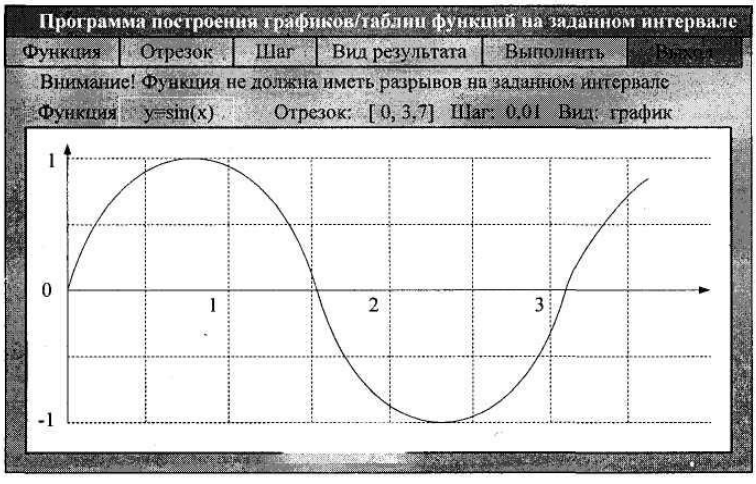

Рис. 2.9. Зовнішній вигляд вікна програми

Очевидно, що в цьому випадку забезпечується гнучкіше управління для користувача, оскільки фактично передбачені такі сценарії роботи: Введення функції – Введення відрізка – Введення кроку – Уточнення виду результату: графік/таблиця – Виведення результату; Зміна відрізка – Виведення результату; Зміна кроку – Виведення результату; Зміна виду результату: графік/таблиця – Виведення результату тощо.

Варіант 3. Інтерфейс з вільною навігацією для цієї програми наведено на рис. 2.10 (природно, обробник цієї події повинен передбачати аналіз даних на повноту і сумісність). Графік будується після натиснення кнопки Побудувати. Змінювати дані можна у будь-який момент і у будь-якому порядку, використовуючи відповідні компоненти введення-виведення [1].

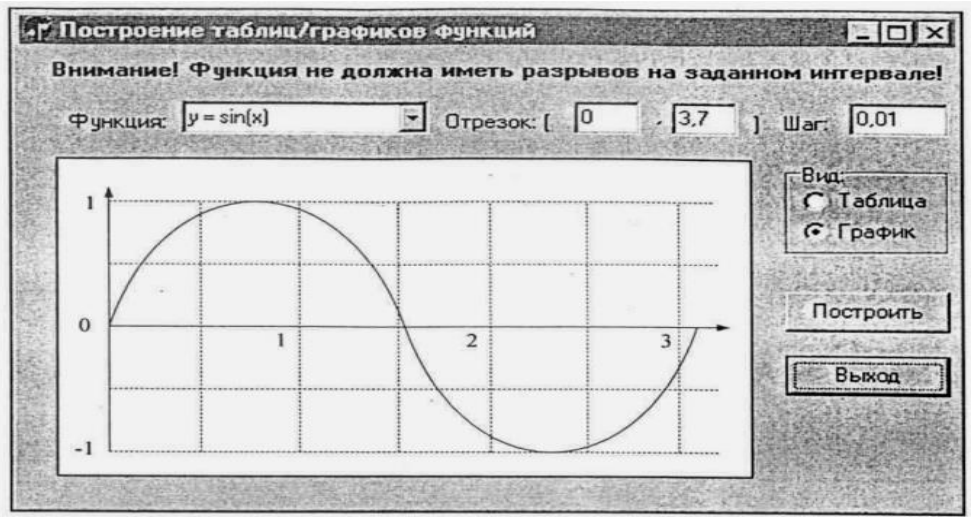

Рис. 2.10. Зовнішній вигляд вікна програми побудови графіків/таблиць функцій: інтерфейс з вільною навігацією

Варіант 4. Інтерфейс прямого маніпулювання для цієї програми подано на рис. 2.11 [1]. Для того, щоб ввести нову формулу, необхідно взяти чистий бланк з теки. Бланк розкривається подвійним натисканням кнопки миші, після чого його необхідно заповнити. Отриманий бланк можна «обрахувати», перенісши на піктограму комп'ютера. Змінювати дані і тип результатів можна у будь-який момент і у будь-якому порядку, «розкривши» бланк. Заповнені бланки, які можуть ще знадобитися, «кладуть» у теку Функції, останні – в «кошик».

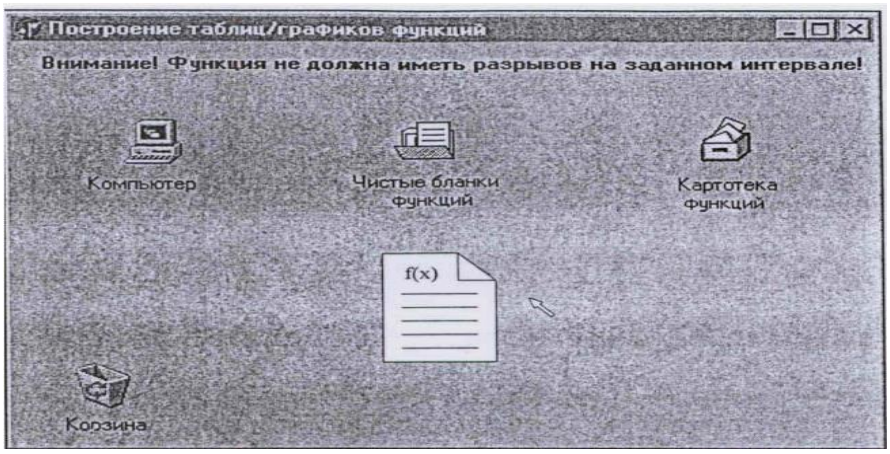

Рис. 2.11. Зовнішній вигляд вікна програми побудови графіків/таблиць функцій: інтерфейс прямого маніпулювання

Контрольні запитання до розділу 2

1. Що розуміють під діалогом між людиною і комп'ютером?
2. Як здійснюється обмін інформацією?
3. Яку модель використовують процедурно-орієнтовані інтерфейси?
4. Яку модель використовують об'єктно-орієнтовані інтерфейси?
5. Який інтерфейс називають примітивним?
6. Що дозволяє користувачеві Інтерфейс-меню?
7. Що дозволяють користувачеві Інтерфейси з вільною навігацією?
8. Що дозволяють користувачеві інтерфейсом прямого маніпулювання?

РОЗДІЛ 3. Елементи методології проектування інтерфейсу

3.1. Психофізичні аспекти взаємодії людини і комп'ютера

При проектуванні користувацьких інтерфейсів необхідно враховувати психофізичні особливості людини, пов'язані зі сприйняттям, запам'ятовуванням і обробкою інформації, центром яких є мозок людини. Принципи роботи мозку людини досліджує когнітивна психологія.

Звернемося до аналізу особливостей роботи центрів мозку людини, пов'язаних з вищеназваними функціями. Для цього розглянемо спрощену інформаційно-процесуальну модель мозку, подану на рис. 3.1 [1].

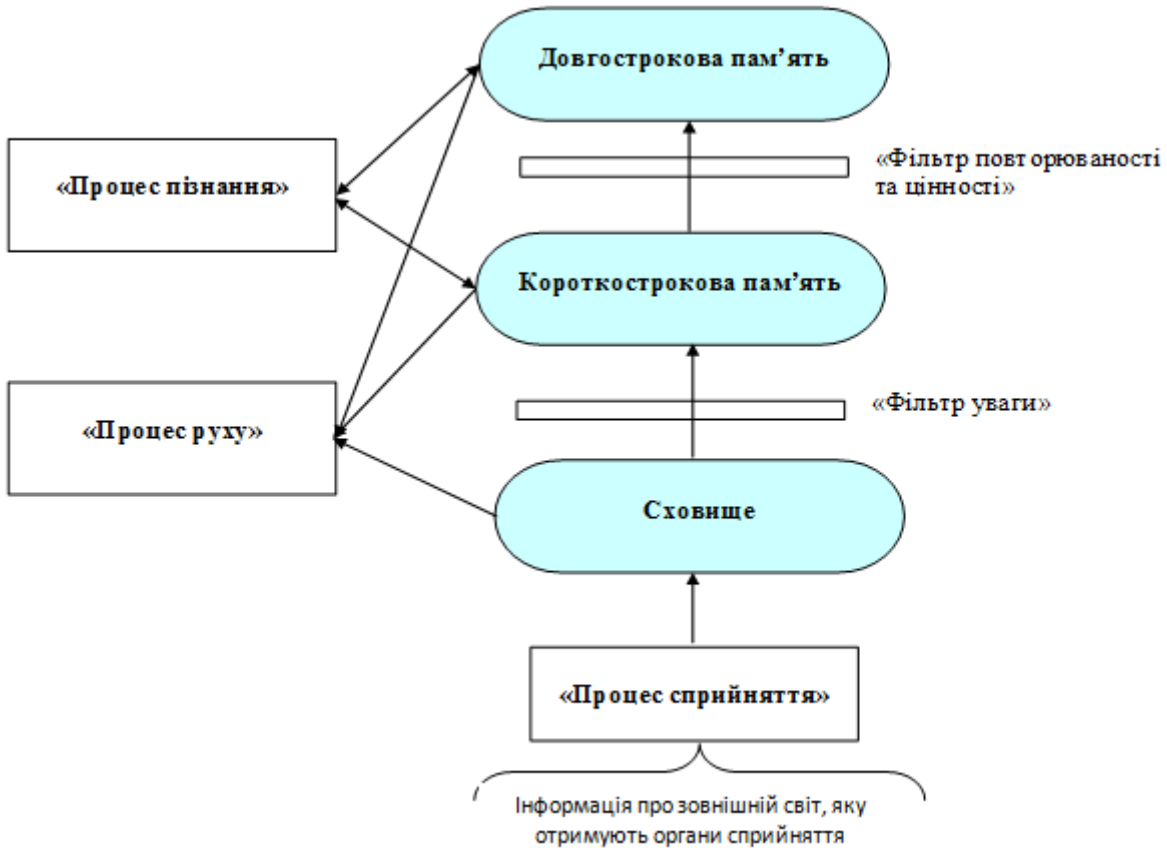

Рис. 3.1. Спрощена інформаційно-процесуальна модель мозку

Інформація про зовнішній світ надходить у мозок у великому обсязі. Ділянка мозку, яку умовно можна назвати «процесором сприйняття», постійно без участі свідомості

переробляє її, порівнюючи з минулим досвідом, і поміщає в сховище вже у вигляді зорових, звукових та інших образів. Будь-які раптові або просто значущі для людини зміни в оточенні привертають увагу. Інформація надходить у короткочасну пам'ять. Якщо ж увага не була привернута, то інформація в сховищі зникає, заміщаючись наступними порціями.

У кожен момент часу фокус уваги може фіксуватися в одній точці. Тому якщо виникає необхідність «одночасно» відстежувати декілька ситуацій, то зазвичай фокус переміщається з одного відстежуваного елемента на інший. Водночас увага «розосереджується», і якісь деталі можуть бути упущені [1].

Істотно і те, що сприйняття багато в чому основане на мотивації. Наприклад, якщо людина голодна, то вона насамперед помічатиме все їстівне, а якщо втомлена, то, увійшовши до кімнати, вона насамперед побачить диван або ліжко.

Необхідно також ураховувати, що в процесі переробки інформації мозок порівнює дані, що надходять, з раніш отриманими. Так, якщо показати людині послідовність символів: A, D, C, 13, то вона може прийняти 13 за B.

Короткострокова пам'ять − «найвужче» місце «системи обробки інформації» людини. Її ємкість приблизно дорівнює 7 ± 2 незв'язаних об'єктів.

Короткострокова пам'ять є свого роду *оперативною пам'яттю* мозку.

Саме з нею працює процесор пізнання, але не затребувана інформація зберігається в ній не більше 30 с. Щоб не забути яку-небудь важливу інформацію, зазвичай повторюють її «про себе», «оновлюючи» інформацію в короткостроковій пам'яті.

При зміні кадру мозок на деякий час блокується: він «освоює» нову картинку, виділяючи найбільш істотні деталі. А отже, якщо необхідна швидка реакція користувача, то різко змінювати картинку не варто [1].

Таким чином, при проектуванні користувацьких інтерфейсів слід мати на увазі, що переважній більшості людей важко, наприклад, запам'ятати і ввести на іншому екрані число, що містить більше 5 цифр, або деяке поєднання букв [1, 2].

Людина вносить до кожної *діяльності* своє *розуміння* того, як вона повинна виконуватися. Це розуміння – *модель діяльності*, що базується на минулому досвіді людини.

Безліч таких моделей зберігається в довготривалій пам'яті людини. У довготривалу пам'ять записуються постійно повторювані відомості або інформація, пов'язана із сильними емоціями.

Довготривала пам'ять людини − сховище інформації з *необмеженою ємкістю і часом зберігання.*

Проте доступ до цієї інформації дуже непростий: імовірно, механізми добування інформації з пам'яті мають асоціативний характер.

Спеціальна методика запам'ятовування інформації (*мнемоніка*) використовує саме цю властивість пам'яті: для запам'ятовування інформації її «прив'язують» до тих даних, які пам'ять вже зберігає і дає змогу легко отримати.

Оскільки доступ до довготривалої пам'яті *утруднений*, доцільно розраховувати не на те, що користувач пригадає потрібну інформацію, а на те, що він її дізнається. Саме тому інтерфейс типу меню так широко використовується.

Особливості сприйняття кольору. Колір у свідомості людини асоціюється з емоційним фоном.

Відомо, що *теплі кольори*: червоний, оранжевий, жовтий *людину збуджують*, а *холодні*: синій, фіолетовий, сірий – *заспокоюють*.

Причому *колір* для людини є *дуже сильним подразником*, тому застосовувати кольори в інтерфейсі необхідно *дуже обережно*.

Слід мати на увазі, що велика кількість відтінків привертає увагу, але швидко стомлює. Тому не варто яскраво розфарбовувати вікна, з якими користувач довго працюватиме. Необхідно враховувати й індивідуальні особливості сприйняття кольорів людиною.

Особливості сприйняття звуку. У інтерфейсах звук зазвичай використовують з різними цілями: для *залучення уваги*, як *фон*, що забезпечує деякий стан користувача, як *джерело додаткової інформації* тощо.

Застосовуючи звук, слід ураховувати, що *більшість людей дуже чутлива до звукових сигналів*, особливо, якщо останні вказують на наявність помилки.

Тому при створенні звукового супроводу доцільно *передбачати можливість його відключення*.

Суб'єктивне сприйняття часу. Людині властиве суб'єктивне сприйняття часу. Вважають, що внутрішній час пов'язаний із швидкістю та кількістю сприйнятої й оброблюваної інформації. Зайнята людина зазвичай часу не помічає.

Скоротити час очікування можна, зайнявши користувача, але не відвертаючи його від роботи. Наприклад, можна надати йому яку-небудь інформацію для аналізу. По можливості доцільно виводити користувачеві проміжні результати: по-перше, він буде зайнятий їх обдумуванням, по-друге, за ними він зможе оцінити майбутні результати і відмінити операцію, якщо вони його не задовольняють.

Щоб зменшити роздратування, що виникає при очікуванні, необхідно дотримуватися основного правила:

інформувати користувача, що замовлені ним операції потребують деякого часу виконання.

Зазвичай для цього використовують індикатори залишку часу: анімовані об'єкти як в інтернеті, зміна форми курсора миші на пісочний годинник. Дуже важливо точно позначити момент, коли система готова продовжувати роботу. Зазвичай для цього використовують помітні зміни зовнішнього вигляду екрана.

3.2. Визначення змісту програмної моделі інтерфейсу

Є три абсолютно різні моделі користувацького інтерфейсу: модель програміста, модель користувача і програмна модель [1]. На рис. 3.2 наведено характеристики моделі користувацького інтерфейсу.

З погляду здорового глузду хорошим слід вважати інтерфейс, при роботі з яким користувач отримує саме те, що він чекав. Уявлення користувача про функції інтерфейсу можна описати у вигляді користувацької моделі інтерфейсу.

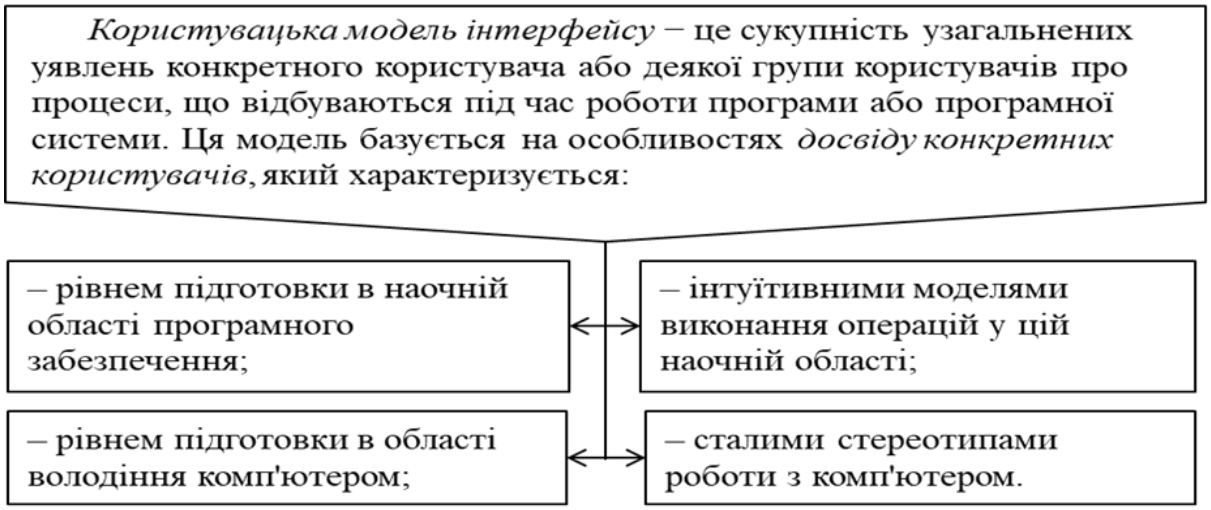

Рис. 3.2. Характеристики моделі користувацького інтерфейсу

Для побудови користувацької моделі необхідно вивчити перераховані вище особливості досвіду очікуваних користувачів програмного забезпечення. З цією метою використовують опитування, тести і навіть фіксують на плівку послідовність дій, здійснюваних у процесі виконання деяких операцій. Приведення у відповідність моделей користувача і програміста, а також побудова на їх базі програмної моделі інтерфейсу завдання не тривіальне (рис. 3.3) [1].

При створенні програмної моделі інтерфейсу слід мати на увазі, що змінити призначену для користувача модель непросто. Підвищення професійного рівня користувачів і їх підготовки в області володіння комп'ютером у компетенцію розробників програмного забезпечення не входить, хоча часто грамотно побудований інтерфейс, який адекватно відображає суть процесів, що відбуваються, сприяє зростанню кваліфікації користувачів [1].

Інтуїтивні моделі виконання операцій у наочній області повинні стати основою для розробки інтерфейсу, а тому в більшості випадків їх необхідно не змінювати, а уточнювати та удосконалювати. Саме небажання або неможливість застосування інтуїтивних моделей виконання операцій приводить до створення штучних надуманих інтерфейсів, які негативно сприймаються користувачами

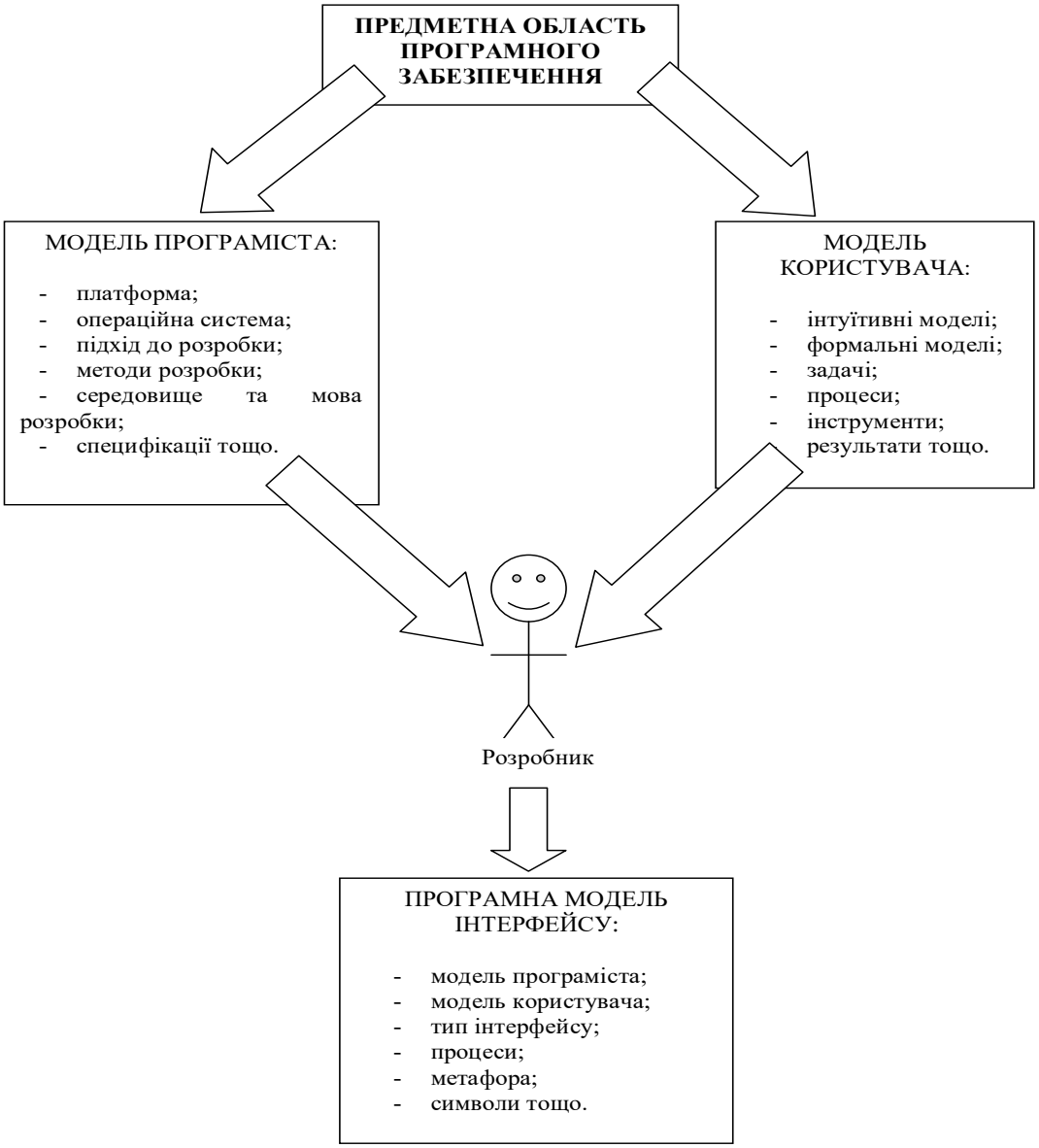

Рис. 3.3. Модель процесу проектування інтерфейсу

3.3. Зміст критеріїв оцінки інтерфейсу користувачем

На рис. 3.4 наведено характеристику критеріїв оцінки інтерфейсів користувачем.

Причому для користувачів-професіоналів, що постійно працюють з тим самим пакетом, на перше місце достатньо швидко виходять другий і третій критерії, а для користувачів-непрофесіоналів, які працюють з програмним забезпеченням періодично і які виконують порівняно нескладні завдання, – перший і третій.

Рис. 3.4. Характеристика критеріїв оцінки інтерфейсів користувачем

З цієї точки зору на сьогоднішній день найкращими характеристиками для *користувачів-професіоналів* є інтерфейси з *вільною навігацією*, а для *користувачів-непрофесіоналів* – інтерфейси *прямого маніпулювання* [1].

Давно відомо, що при виконанні операції копіювання файлів за інших рівних умов більшість професіоналів використовують оболонки типу Far, а непрофесіонали – «перетягування об'єктів» Windows.

3.4. Класифікації типів діалогів та їх форм

Діалог – це процес обміну інформацією між користувачем і програмною системою, здійснюваний через *інтерактивний термінал* і за *певними правилами*. Доречно розглянути типи діалогів з погляду поняття «користувацький інтерфейс» [1]. На рис. 3.5 наведено характеристику типів діалогів.

Форми діалогу. Ніякий діалог неможливий, якщо не існує мови, зрозумілої «співбесідникам». Опис мови, якою ведеться діалог, містить визначення її синтаксису – правил, що визначають допустимі конструкції (слова, речення) мови або її форму, і семантики – правил, що визначають значення синтаксично коректних конструкцій мови або її зміст [1].

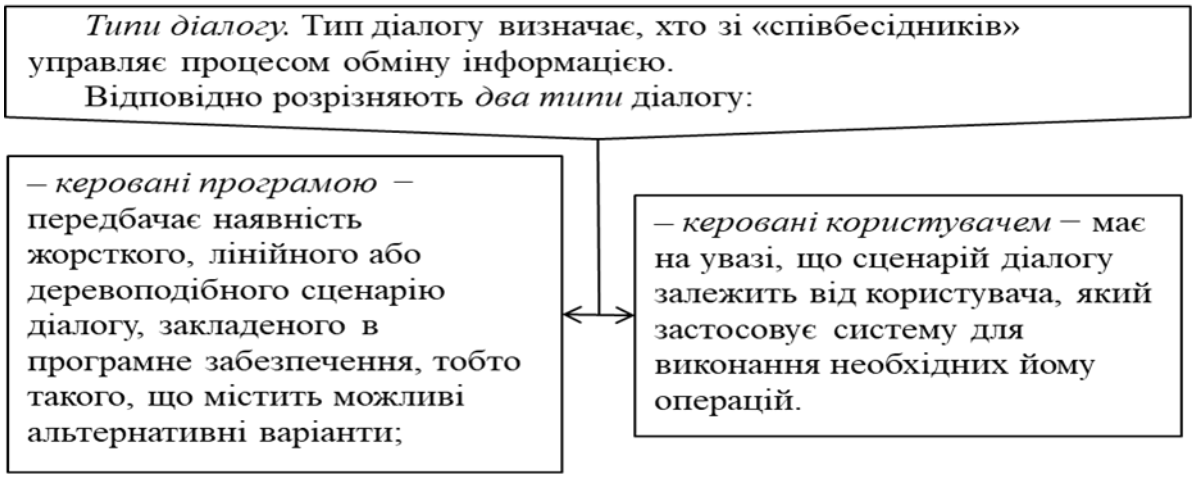

Рис. 3.5. Характеристика типів діалогів

На рис. 3.6 наведено характеристику форм діалогів.

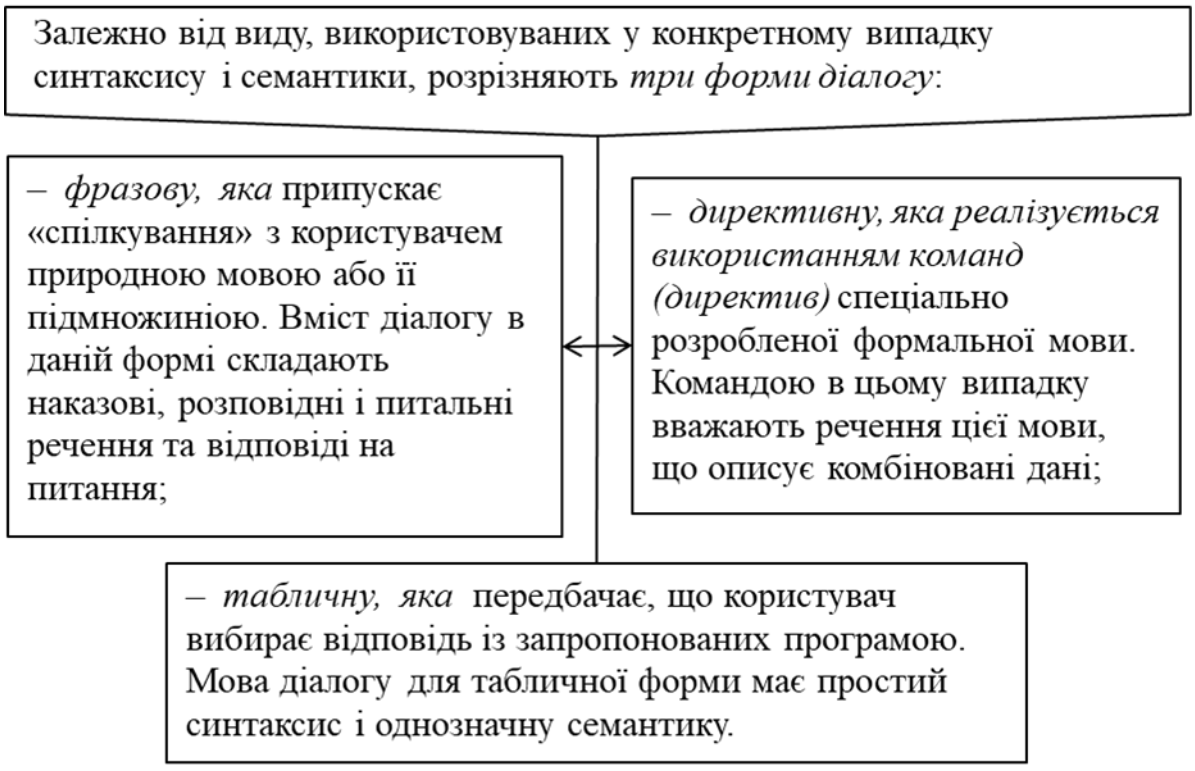

Рис. 3.6. Характеристика форм діалогів

Інтерфейс, що реалізовує фразову форму діалогу, повинен перетворювати повідомлення з природно-мовної форми у форму внутрішнього уявлення і, навпаки, виконувати аналіз і синтез повідомлень користувача і системи, відстежувати та запам'ятовувати попередню частину діалогу [1].

Основна *перевага фразової форми* полягає у відносно вільному спілкуванні із системою.

Основними *недоліками фразової форми* при використанні підмножини природної мови є:

- великі витрати ресурсів;
- відсутність гарантії однозначної інтерпретації формулювань;
- необхідність введення довгих граматично правильних фраз.

Директивна форма реалізується використанням команд (директив) спеціально розробленої формальної мови. Командою в цьому випадку вважають речення цієї мови, що описує комбіновані дані, яке містить ідентифікатор процесу, що ініціюється, і, якщо є потреба, дані для нього.

Команду можна вводити:

- у *вигляді рядка тексту*, спеціально розробленого формату, наприклад, команди MS DOS, які вводяться в командному рядку;
- *натисненням деякої комбінації клавіш клавіатури*, наприклад, комбінації «швидкого доступу» сучасних Windows-додатків;
- за допомогою *маніпулювання мишею*, наприклад, «перетягуванням» піктограм;
- *комбінацією другого і третього* способів.

Основними *перевагами директивної форми* є:

- порівняно невеликий обсяг інформації, що вводиться;
- гнучкість − можливості вибору операції в цьому випадку обмежені тільки набором допустимих команд;
- орієнтація на діалог, керований користувачем;
- використання мінімальної області екрана або невикористання її взагалі;
- можливість поєднання з іншими формами.

Недоліки директивної форми:

- ✓ практична відсутність підказок на екрані, що вимагає запам'ятовування команд, які вводяться, і їх синтаксису;
- ✓ майже повна відсутність зворотного зв'язку про стан ініційованих процесів;
- ✓ необхідність навичок введення текстової інформації або маніпуляцій мишею;
- ✓ відсутність можливості налаштування користувачем.

Таблична форма передбачає, що користувач вибирає відповідь із запропонованих програмою.

Мова діалогу для *табличної форми* має *простий синтаксис і однозначну семантику*, що достатньо легко реалізувати.

Зручна ця форма і для користувача, оскільки вибрати завжди простіше, ніж пригадати, що особливо істотно для *користувача-непрофесіонала* або користувача, що рідко використовує конкретне програмне забезпечення.

Проте застосування цієї форми можливе не завжди: її можна використовувати тільки тоді, коли існує обмежена кількість відповідей на конкретне питання.

Перевагами табличної форми є:

- наявність підказки, що зменшує навантаження на пам'ять користувача, оскільки ця форма орієнтована не на запам'ятовування, а на вибір;
- скорочення кількості помилок введення: користувач не вводить інформацію, а вказує на неї;
- скорочення часу навчання користувача;
- можливість поєднання з іншими формами діалогу;
- у деяких випадках можливість налаштування користувачем.

До недоліків цієї форми відносять:

- ✓ необхідність наявності навичок навігації на екрані;
- ✓ використання досить великої площі екрана для зображення візуальних компонентів;
- ✓ інтенсивне використання ресурсів комп'ютера, що пов'язане з необхідністю постійного оновлення інформації на екрані.

Слід мати на увазі, що типи і форми діалогу вибирають незалежно один від одного: будь-яка форма може бути застосовна для обох типів діалогів (рис. 3.7) [1]. Проте фразова форма, яка використовується в діалозі, керованому користувачем, як правило, припускає складніший синтаксис і семантику мови діалогу, оскільки програма повинна «розуміти» користувача.

Складне програмне забезпечення зазвичай взаємодіє з користувачем за допомогою діалогів різних типів і форм залежно від поставлених завдань. Причому, окрім діалогів, що відбуваються в процесі нормальної роботи програмного

забезпечення, що називаються синхронними, передбачають діалоги, що виникають за ініціативою системи або користувача при порушенні сценарію нормального процесу. Такі діалоги називають асинхронними. Зазвичай їх використовують для видачі екстрених повідомлень від системи або користувача.

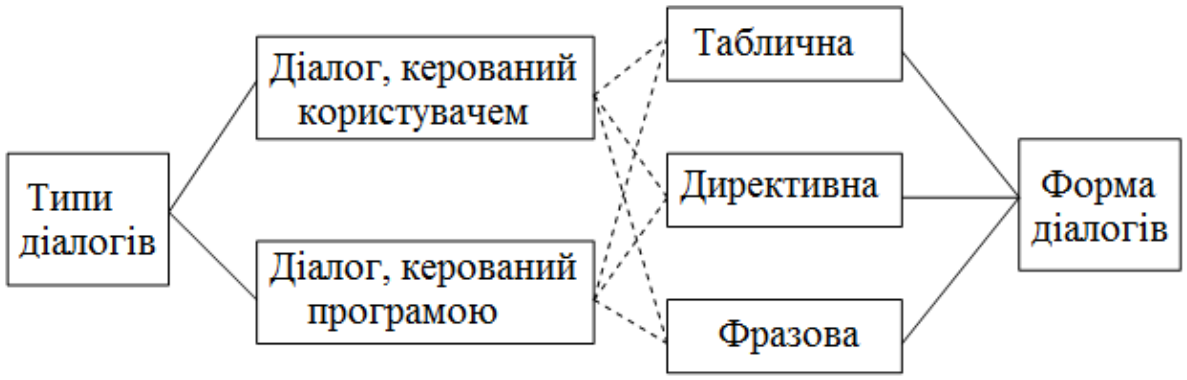

Рис. 3.7. Відповідність типів діалогів і його форм

3.5. Проектування діалогів

На рис. 3.8 наведено склад стадій проектування реалізації діалогів.

В основу проектування абстрактних діалогів повинна закладатися ідеологія технологічного процесу, для автоматизації якого призначається програмний продукт. Саме аналізуючи складові технологічного процесу, що автоматизується, розробник визначає сценарії діалогів, які мають бути передбачені в програмному забезпеченні. Окрім сценаріїв, при проектуванні абстрактних діалогів використовують діаграми стану інтерфейсу або графи діалогу [1].

Граф діалогу − орієнтований зважений граф, кожній *вершині* якого співвіднесена *конкретна картинка на екрані* (кадр) або певний стан діалогу, що характеризується набором доступних користувачеві дій.

Дуги, що виходять з вершин, *показують можливі зміни станів* при виконанні користувачем указаних дій.

Умови переходів із стану в стан і операції, що виконуються при цьому, указують *ваги* дуг.

Рис. 3.8. Склад стадій проектування реалізації діалогів

Кожен маршрут на графі відповідає можливому варіанту діалогу. Причому подання діалогу у вигляді графа, залежно від стадії розробки, може виконуватися з різним ступенем деталізації.

По суті граф діалогу – це граф станів кінцевого автомата, що моделює поведінку програмного забезпечення при діях користувача. Для подання таких графів використовуються дві нотації: нотація діаграм станів структурного підходу до розробки (рис. 3.9) і нотація діаграм станів UML (рис. 3.10) [1].

Нотація – це система умовних письмових позначень, прийнята в якій-небудь галузі людської діяльності.

Причому нотація UML є більш потужною, оскільки дає змогу використовувати узагальнені стани. Тому, щоб не вводити нову нотацію для подання графа діалогу, використовують позначення UML.

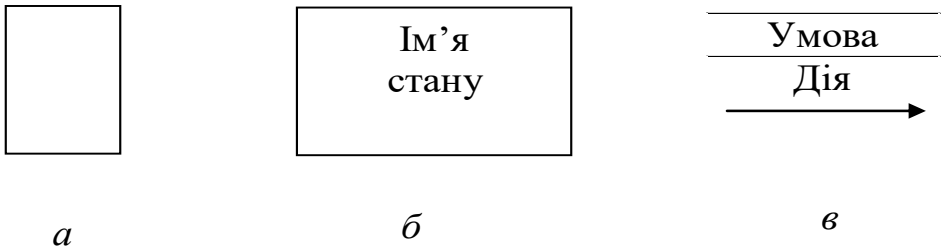

Рис. 3.9. Умовне позначення діаграм переходів станів: а – термінальний стан; б – проміжний стан; в – перехід

Ім’я класу
Атрибути
Операції ( )
Відповідальність

Рис. 3.10. Повне умовне позначення класу UML

Розглянемо реалізацію побудови графів діалогу на конкретному прикладі.

Приклад. Розробити граф діалогу для системи вирішення комбінаторно-оптимізаційних задач.

Оскільки діалог на верхньому рівні повинен забезпечувати реалізацію діаграми варіантів використання, початковий варіант графа діалогу будуємо на основі аналізу цієї діаграми (рис. 3.11) [1].

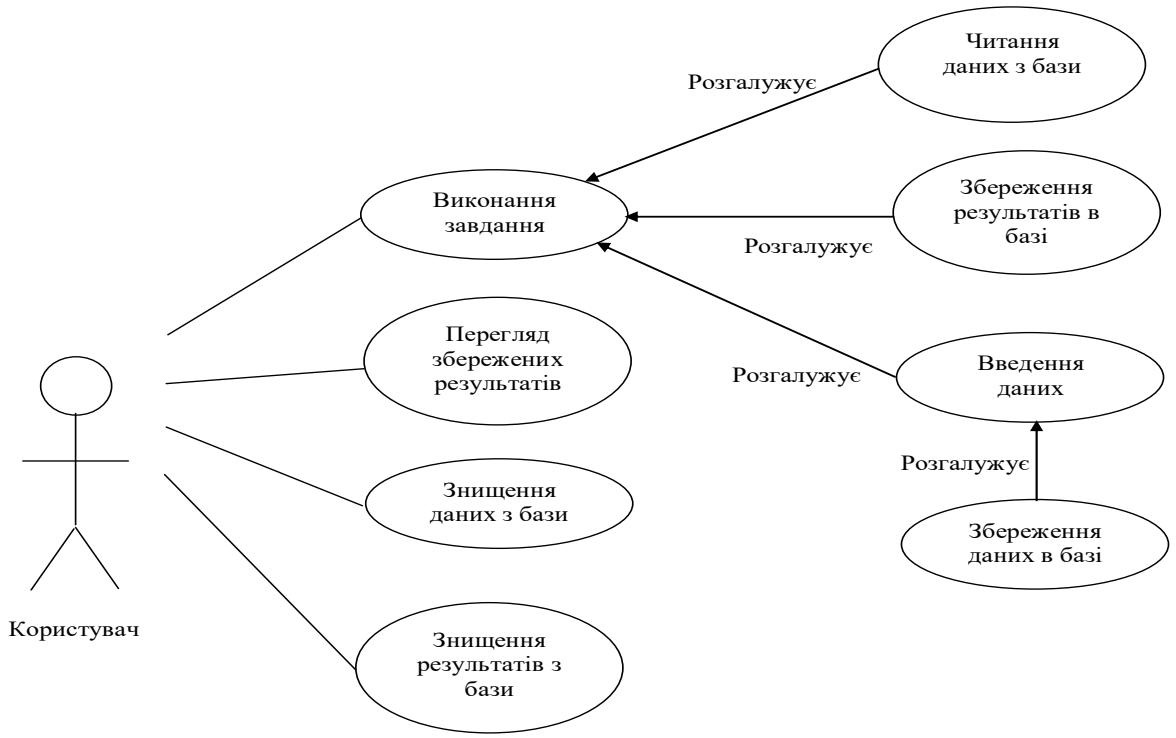

Рис. 3.11. Діаграма варіантів використання для системи рішення комбінаторно-оптимізаційних задач

Можна припустити, що користувач ухвалюватиме рішення про збереження або видалення результатів після їх перегляду. Тому ці операції природно об'єднати в єдину групу. Крім того, в

ту ж групу доцільно додати операцію друку результатів. Аналогічно перегляд даних доцільно об'єднати з їх видаленням або коректуванням. Операцію Нове завдання доцільно помістити в окрему групу (рис. 3.12) [1].

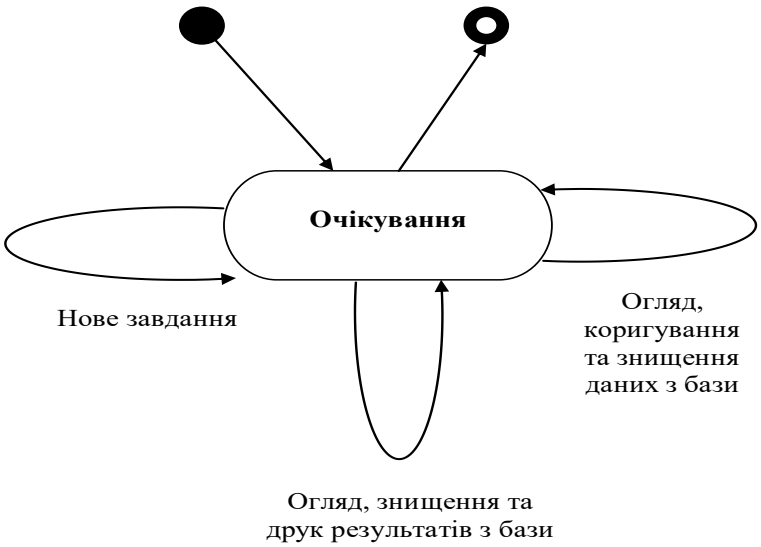

Рис. 3.12. Граф абстрактного діалогу системи рішення комбінаторно-оптимізованих задач

На верхньому рівні діалогом очевидно повинен управляти користувач. Директивна і таблична форми можуть використовуватися альтернативно, за бажанням користувача, а застосування фразової форми недоцільне [1].

Приклад. Деталізувати діалог Нове завдання [1].

На базі вищенаведеного сценарію *Виконання завдання* можна запропонувати граф діалогу, керованого системою (рис. 3.13, а). Проте цей же діалог можна подати і у вигляді діалогу, керованого користувачем (рис. 3.13, б).

Аналіз графів діалогу свідчить: діалог, керований системою, у цьому випадку сильно обмежує користувача у виборі варіантів дії; діалог, керований користувачем, припускає вибір дії після кожного кроку, хоч за змістом ці кроки найчастіше виконуватимуться послідовно. Тому для реалізації краще використовувати комбінований варіант діалогу, який ураховує наявність сценарію, але допускає відхилення від нього за бажанням користувача (рис. 3.14) [1].

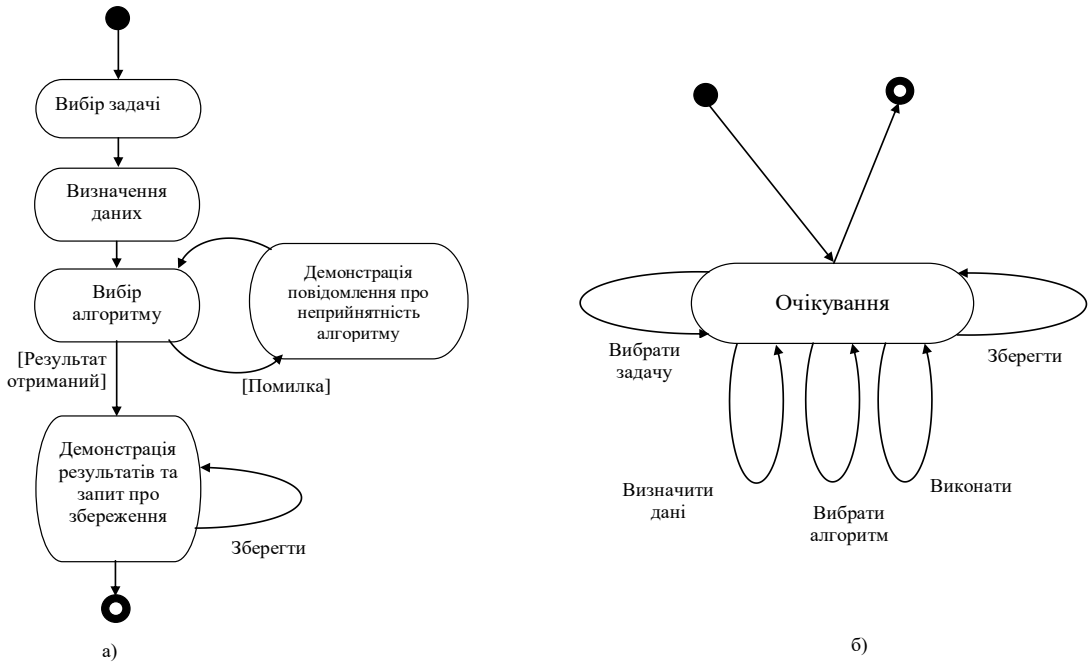

Рис. 3.13. Графи абстрактного діалогу. Нове завдання:
а – діалог, що управляється системою; б – діалог,
що управляється користувачем

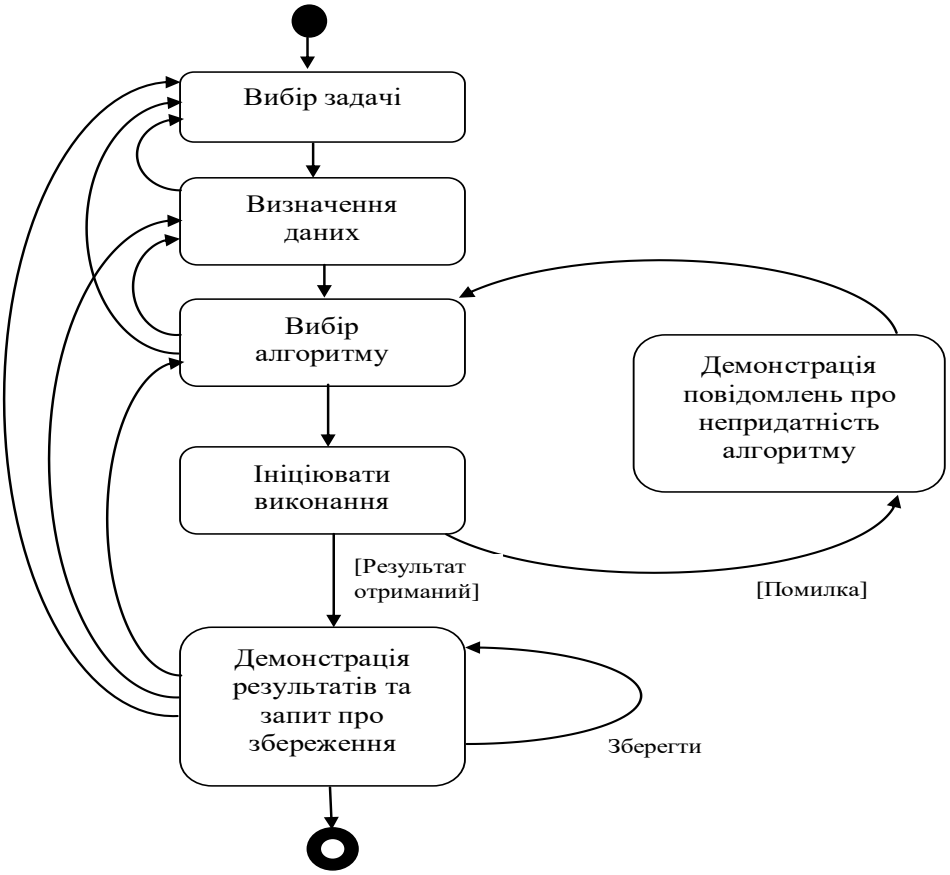

Рис. 3.14. Граф абстрактного діалогу комбінованого типу

Далі необхідно визначити, які форми діалогу можна використовувати для кожного кроку діалогу. Перший крок – Вибір завдання, який включає три варіанти, тому має сенс використовувати табличну форму. Другий крок – Визначення даних поки ще не конкретизований, отже, визначити його форму ще не можливо. Третій крок – Вибір алгоритму знову ж таки передбачає вибір, причому кількість варіантів невелика: доцільно використовувати табличну форму. У решті випадків також переважною виявляється саме ця форма.

Останній етап проектування інтерфейсів – розробка конкретних операцій введення-виведення для кожного діалогу з урахуванням специфіки форми інтерфейсу.

Контрольні запитання до розділу 3

1. Охарактеризуйте психофізичні передумови взаємодії людини і комп’ютера.
2. Наведіть опис програмної моделі користувацького інтерфейсу.
3. Наведіть критерії оцінки інтерфейсу користувачем.
4. Наведіть визначення типів діалогів.
5. Наведіть визначення форми діалогів.
6. Наведіть алгоритм розробки діалогів.

РОЗДІЛ 4. Моделювання поведінки користувачів, робочого середовища та вирішуваних ними задач

4.1. Моделювання поведінки користувачів та їх робочого середовища

Одна з головних причин створення невдалих продуктів полягає в недостатньому залученні користувачів до проекту. Не менше значення мають і наслідки недостатньо активної участі користувачів у проекті, зокрема відсутність глибоких знань про реальних користувачів продукту, середовище використання розробки та рівня застосування тих чи інших задач [1].

Деякі продукти відрізняються досить конкретним колом користувачів та досить певним середовищем використання і задачами. Для інших проектів сукупність користувачів відрізняється великим розмаїттям поряд із значними відмінностями в середовищах і задачах.

Детальна інформація про користувачів, середовища і задачі допомагає встановити рамки, у яких повинно здійснюватись проектування КІ продукту та забезпечуватись його практичність.

Поряд з вимогами до КІ та практичності критична інформація про користувачів, середовища і задачі допомагає колективу розробників виділити ті особливості продукту, які бажають бачити реальні користувачі або яких вони потребують. Інформація про користувачів та їх потреби важлива для відповідного вибору методів створення КІ та підходів до проектування спільного стилю додатків [1].

Приклади найбільш помітних *характеристик користувача* у зв'язку з його роботою на комп'ютері і з його ставленням до ПЗ:

- *попередні знання та досвід*, використовувані *в області апаратного та програмного забезпечення*;
- *попередні знання та досвід*, які стосуються *підтримки задач прикладної області*;
- *фізичні та індивідуальні* характеристики;
- *фізичні та соціальні особливості* робочого середовища.

Усім користувачам догодити неможливо. Програмний продукт та його КІ не можуть повністю підходити всім користувачам. Не потрібно висувати надто багато припущень стосовно до будь-якої

групи користувачів, а також фізичних та соціальних умов для роботи [1].

Критична інформація про користувачів містить *статистичні розподіли* для таких *характеристик*:

- *освіта, знання та досвід* роботи з комп'ютерними системами й ПЗ, а також у прикладній області, для підтримки якої розробляється ПЗ;
- *ставлення до роботи*, бізнес-процедур та виконуваних задач;
- *ставлення до підтримки* апаратного та програмного забезпечення комп'ютерів;
- *характеристики соціального та фізичного робочого середовища;*
- *фізичні та особистісні характеристики*.

4.2. Методи збирання та аналізу інформації про користувачів

Основна мета аналізу полягає в тому, щоб подивитись, що відбувається в робочому середовищі, і визначити, у чому може полягати реальна допомога з боку програмної системи [1].

Методи залучення користувачів у проект:

- ✓ *метод спостереження* – спостереження за реальними користувачами, які виконують реальну роботу;
- ✓ *метод опитування* – опитуються всі учасники проекту, які виконують комплекс задач або входять у робоче середовище;
- ✓ *метод індивідуальних інтерв'ю* – неформальні інтерв'ю з реальними користувачами;
- ✓ *метод групового інтерв'ю* – формальні або неформальні інтерв'ю з колективом або групою реальних користувачів;
- ✓ *інші джерела інформації*.

На рис. 4.1 наведено перелік питань, які мають бути сформовані у вигляді тестів.

Аналіз слід продовжувати не перервно, постійно уточнюючи його висновки та результати.

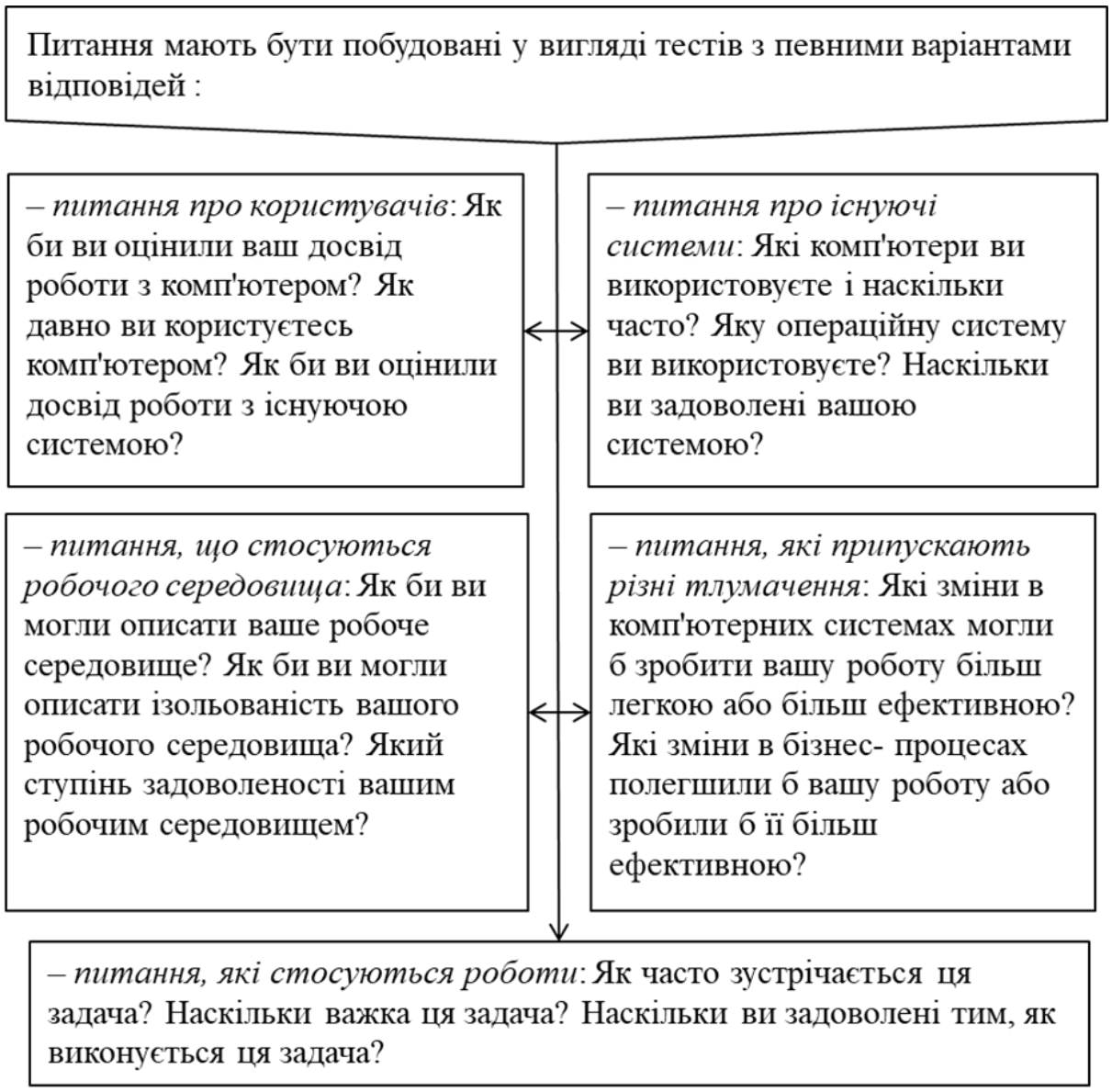

Рис. 4.1. Склад та зміст питань, які мають бути сформовані у вигляді тестів

4.3. Елементи етапу концептуального проектування інтерфейсу

Усі задачі, пов'язані з проектуванням та розробкою КІ, а також забезпеченням практичності продукту, надзвичайно важливі.

Перша конкретна проектна задача – *концептуальна.* Вона передбачає вибір *правильних основних положень для прийняття проектних рішень* [1].

У багатьох відношеннях *концептуальний проект КІ* – це, по суті, його *архітектура*.

Послідовні рівні проектних рішень дають змогу нарощувати визначення і ступінь деталізації зовнішнього вигляду, поведінки, інформаційної підтримки та користувацької взаємодії на більш пізніх етапах проектування.

Усі кроки проектування в рівному ступені важливі. Концептуальне проектування, яке стосується концептуальних моделей, стилю КІ та навігаційних структур, не зупиняється на деталях зовнішнього вигляду, поведінки, взаємодії і на функціональних можливостях продукту [1].

Перший правильний крок важливий для збільшення шансів на найменш короткий та безболісний шлях до досягнення цілей проекту стосовно КІ, практичності, узгодженості, інформаційної підтримки, часу відгуку, надійності та інших проектних чинників.

З одного боку, альтернативні концептуальні рішення формулюються, оцінюються і піддаються ітеративному перегляду, а також дуже швидкому нарощуванню при невеликих витратах – на концептуальному рівні. З другого – помилка в концептуальних проектних рішеннях, не помічена на ранніх проектних етапах розробки, передається на наступний етап і потенційно впливає на множину екранів, взаємодій, підтримку користувачів, програмний код, навчання, тестування, план-графік та вартість розробки. Вартість виправлення такої помилки буде тим вищою, чим довше помилка залишається невиявленою [1].

На основі аналізу характеристик користувачів, вимог, задач та ділових потреб можна представити в явній і наочній формі основоположну ідею проекту стосовно візуалізації та взаємодії, організаційні принципи та структуру, а також типові поведінкові моделі ПЗ.

Концептуальний проект являє собою попередній ескіз, який показує основні функції КІ, візуальні, інформаційні та інші характеристики проекту продукту. Архітектура КІ – це буквально

розподіл роботи між системою та користувачем, а також програмний інтерфейс між ними.

Альтернативи, закладені в концептуальний проект, *одержують наочність у специфікації та прототипі*.

Для одержання *початкової інформації* від користувачів застосовуються методи *залучення користувачів до участі в проекті*.

Для *прогнозування перспективних шляхів розвитку проекту* застосовуються методи оцінки практичності.

На рис. 4.2 наведено основні характеристики концептуального проекту.

Рис. 4.2. Зміст основних компонентів концептуального проекту

Найбільш важливий аспект концептуального проекту – його потенціал з точки зору формування і бачення.

Концептуальний проект установлює те, що називається «*враженням і відчуттям»* від програмного продукту, а також дає змогу виявити, що із спроектованого та сконструйованого під «обгорткою» проекту служить для підтримки *бачення*.

Бачення задає відчуття того, наскільки продукт *логічно узгоджений* та працює на підтримку користувацьких задач.

На рис. 4.3 наведено склад шляхів формування бачення при розробці концептуального проекту.

На рис. 4.4 наведено склад ітерацій при розробці концептуального проекту.

Рис. 4.3. Склад шляхів формування бачення при розробці концептуального проекту

Рис. 4.4. Склад ітерацій при розробці концептуального проекту

На рис. 4.5 наведено зміст основної задачі, яка пов’язана з концептуальним проектуванням.

Рис. 4.5. Зміст основної задачі концептуального проектуванням

При реалізації етапу концептуального проектування виникає додаткова задача формування користувацької моделі. Характеристики цієї моделі наведено на рис. 4.6.

Сприйняття роботи системи користувачем називається користувацькою моделлю.

Це погляд користувача на те, який вигляд мають події, які відбуваються в системі; така модель представляє ефективний спосіб вивчення та прогнозування поведінки ПЗ.

Важливий аспект задач концептуального проектування полягає в посиленні цієї особливості вивчення та поведінки

користувача для свідомого формуванні КІ, щоб стимулювати появу користувацької моделі, яка відповідає намірам проектувальника.

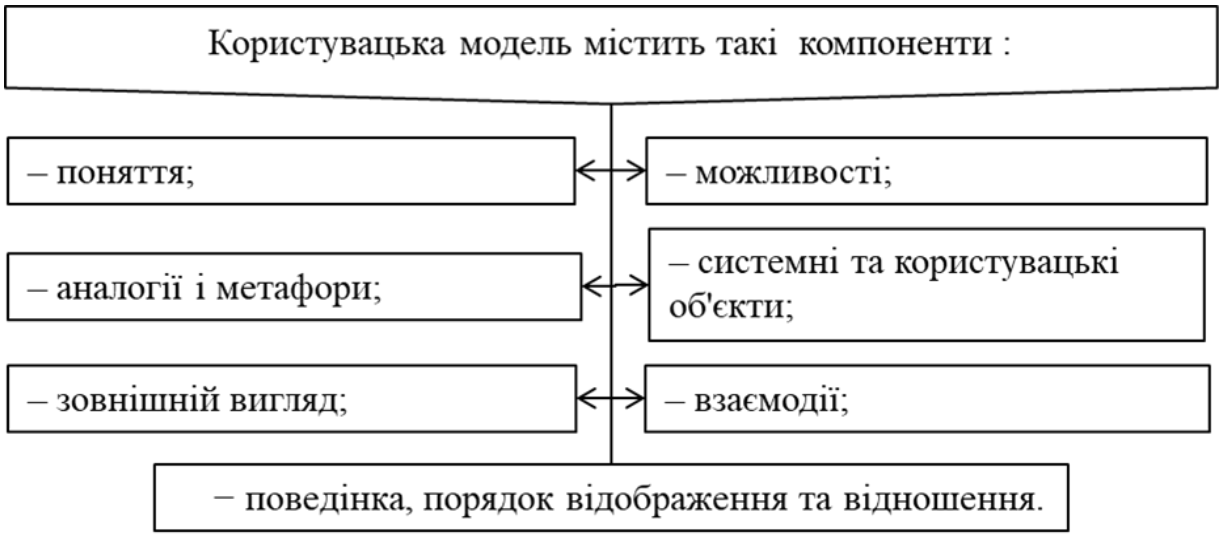

Рис. 4.6. Характеристики користувацької моделі

Зміст критеріїв, яким повинна відповідати модель користувацького інтерфейсу, наведено на рис. 4.7.

Рис.4.7. Зміст критеріїв, яким повинна відповідати модель інтерфейсу

4.4. Розробка макетів, моделей та прототипів інтерфейсів

4.4.1. Визначення складу та змісту макетів, моделей та прототипів інтерфейсів користувача

У багатьох галузях при розробці продуктів поширена практика якомога більш раннього надання їм *наочного вигляду* – те, що в індустрії програмних засобів називається *візуалізацією*.

Створення прототипу чи *прототипування* – термін, яким часто зловживають, який часто вживають неправильно.

Незалежно від використовуваної назви або технічних особливостей *мета проектного колективу полягає у візуалізації проектних рішень*, які приймають одну або декілька форм.

Підхід на основі прототипування можна охарактеризувати за допомогою таких визначень, як *швидкість, точність, можливість повторного використання та скрупульозність*.

Макет – це матеріалізований у вигляді набору *статичних зображень* проект.

Поведінкові та динамічні стани подаються статичними зображеннями. Сукупність статичних зображень, яка демонструє поведінку системи під час прогону сценарію користувацької взаємодії, називається *розкадровкою*.

Модель (імітаційна модель, імітація) являє собою матеріалізацію проекту, вона будується з *використанням засобів* реалізації, *відмінних від тих*, які передбачається використовувати для розробки продукту.

Прототип являє собою матеріалізацію проекту, яка побудована з використанням *передбачуваних засобів розробки* для продукту. Засоби розробки містять *апаратні засоби, операційну систему та мови програмування*.

Практика візуалізації має велике значення також при проектуванні та розробці КІ. Користувачі, ділові спонсори, проектний колектив та інші учасники команди з розробки продукту можуть на ранній стадії його створення поглянути на те, що ж реально виходить на шляху до поставки. Цей перший погляд дає можливість зацікавленим сторонам внести проміжні корективи. Для проектного колективу це ще один метод розробки та можливість перевірити правильність прийнятих рішень, перш ніж вносити зміни буде надто пізно або надто дорого [1].

4.4.2. Визначення цілей візуалізації проекту інтерфейсу користувача

Візуалізація інтерфейсу користувача являє собою цикл поступового уточнення та нарощування проектних рішень у відповідь на питання, які виникають у ході розробки. На ранніх етапах розробки вимоги, проектні концепції та альтернативи досліджуються за допомогою підходів, які не вимагають великих витрат і дають швидку віддачу.

Низькорівневі деталі проекту не піддаються глибокому аналізу. На цих етапах не обов'язково використовуються передбачувані засоби розробки, однак це не сприяє проясненню того, який вигляд матиме і як працюватиме проект стосовно передбачуваного середовища його використання. Використання планованих інструментальних засобів допомагає оцінити потреби в навчанні й часі, необхідні для розробки [1].

Для оцінки спрощених та обмежених представлень складних проектних рішень використовуються евристичні методи і методи сумісної розробки. Від членів проектного колективу та користувачів вимагається подолати концептуальні обмеження, накладені існуючими на папері макетами, розкадровкою, моделями, а також очікувані проектні проблеми та обмеження, пов'язані з використанням цих засобів. Бажано якомога швидше створити спрощений прототип КІ, використовуючи передбачувані засоби розробки і компоненти. Це допомагає прояснити проблеми користувацької взаємодії, пов'язані з логікою, кроками роботи та обмеженнями інструментарію.

Методи прискорення розробки [1]:

– складання прискореного плану-графіка;

– швидкий перехід до прототипування;

– внесення виправлень безпосередньо по ходу проекту;

– припустимість жорсткого програмування;

– концентрація на головних задачах і методах, нехтування неістотними деталями.

При використанні методів візуалізації слід спрямувати основні зусилля на досягнення таких цілей [1]:

– *візуалізація* основних складових, які визначають рівень задоволеності користувачів;

– *увага до дрібниць*, які можуть стати основними перешкодами на шляху до досягнення задоволеності користувачів;

– *оцінка значущості* розглядуваних питань з точки зору проекту.

4.4.3. Метод матеріалізації проекту інтерфейсу користувача

Основна проблема проектного колективу полягає в необхідності прийняти рішення про те, які аспекти проекту підлягають *матеріалізації* з метою їх завчасної оцінки сумісно з користувачами і які методи для цього застосувати. Що стосується великих продуктів, а також аспектів можливостей і практичності, пов'язаних з функціями і КІ, вони вимагають розгляду багатьох питань [1]. На рис. 4.8 наведено характеристики етапу матеріалізації проектних рішень при розробці користувацького інтерфейсу.

Рис. 4.8. Технічні основи матеріалізації проектних рішень

На рис. 4.9 Наведено класифікацію методів візуалізації.

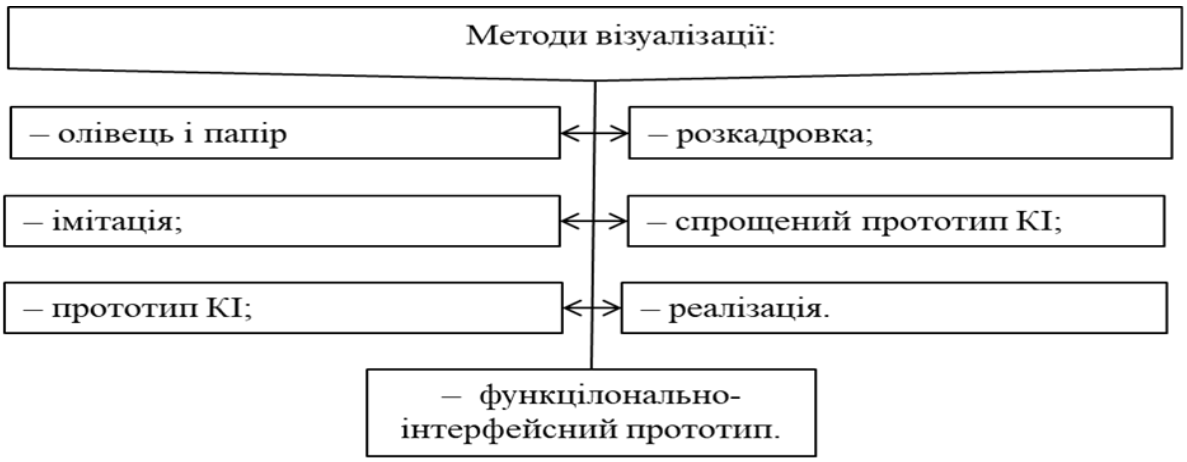

Рис. 4.9. Класифікація методів візуалізації

На рис. 4.10 наведено склад аспектів, які не підлягають матеріалізації до їх реалізації.

Рис. 4.10. Склад аспектів, які не підлягають матеріалізації до їх реалізації

Контрольні запитання до розділу 4

1. Охарактеризуйте користувачів продукту, їх роботу та середовище.
2. Наведіть методи збирання інформації про користувачів.
3. У чому полягає концептуальне проектування?
4. Наведіть визначення макетів, моделей та прототипів інтерфейсів користувача.
5. Наведіть приклади макетів, моделей і прототипів інтерфейсів користувача.
6. Визначте цілі візуалізації проекту.
7. Охарактеризуйте методи матеріалізації проектних рішень.
8. Поясніть процес відкидання прототипів.

РОЗДІЛ 5. Визначення складу та змісту поняття «меню». Проектування меню

5.1. Склад та зміст поняття «меню»

Меню – це метод взаємодії користувача із системою, при якому користувач вибирає із запропонованих варіантів, а не надає системі свою команду. Приклад стандартного меню наведено на рис. 5.1.

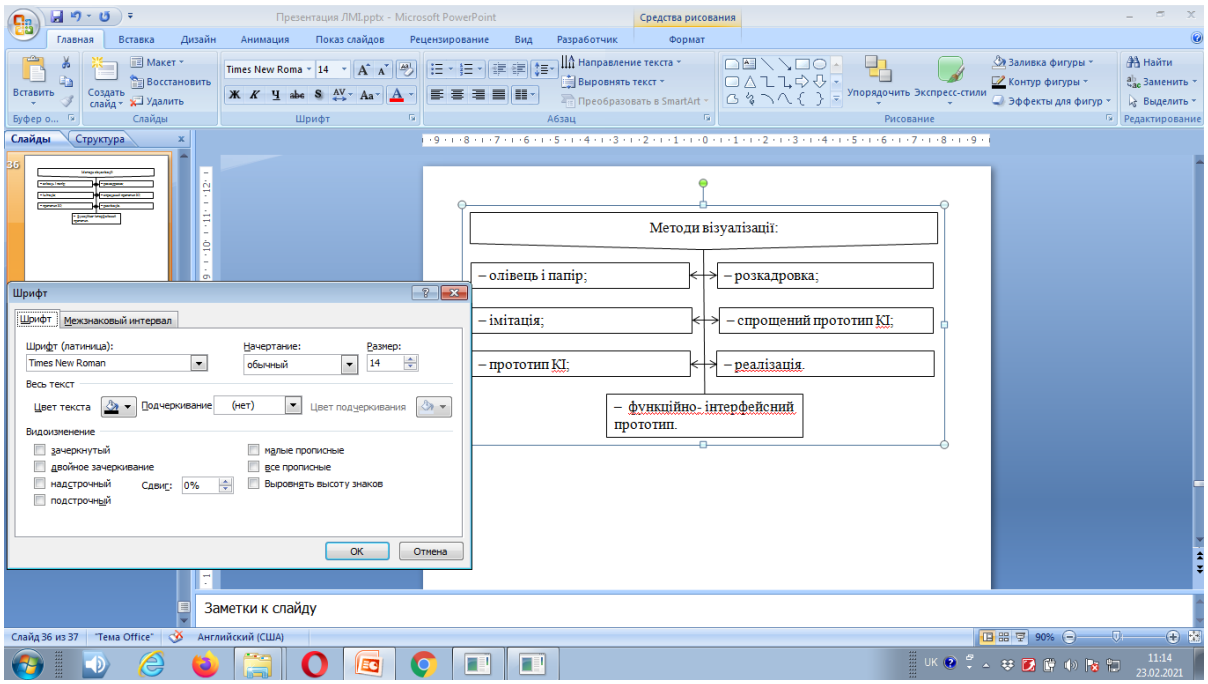

Рис. 5.1. Стандартне випадне меню

Відповідно діалогове вікно з декількома кнопками (і без єдиного поля введення) також є меню (рис. 5.2).

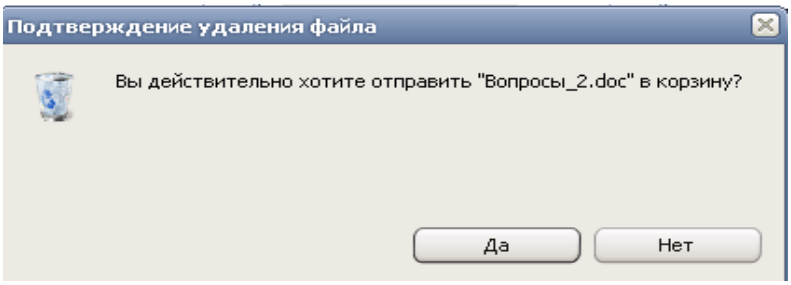

Рис. 5.2. Приклад меню

У більшості систем меню є *об'єктивним благом*, але вони *неефективні* переважно в системах із *зовнішнім середовищем або перебігом часу* [1].

5.2. Визначення типів меню

На рис. 5.3 наведено визначення типів (таксономій) меню, які використовуються в користувацьких інтерфейсах.

Рис. 5.3. Типи (таксономії) меню та їх характеристики

У деяких ситуаціях ці два типи меню можуть зливатися в один. Наприклад, меню, що складається з кнопок доступу до меню, може працювати і як статичні (користувачі натискають на кнопки), і як динамічні (користувачі викликають меню) [1].

Кожен тип меню в кожній з таксономій має певні недоліки. *Статичні меню з першої таксономії*, як правило, забезпечують

високу швидкість роботи, краще навчають користувачів, але займають місце на екрані.

Навпаки, з *динамічними меню ситуація протилежна*. У другій таксономії перший тип (меню, що розгортаються в просторі) забезпечує велику підтримку контексту дій користувачів, але ця підтримка варта втрати екранного простору. Другий тип раціональніше використовує простір, але гірше підтримує контекст [1].

Втім, реальність для таксономій є іншою. Наприклад, майстер, будучи і динамічним меню з першої таксономії, і меню, що розгортається в часі, з другої, не є швидшим, ніж, наприклад, меню, що розкривається. Тому дуже корисно навчитися аналізувати вплив і взаємопроникнення різних типів меню, а також усвідомлювати їх місце в інтерфейсі.

5.3. Проектування структури меню

5.3.1. Будова окремих елементів меню

Ефективність меню найбільше залежить від будови окремих елементів і їх взаємного зв’язку (угрупування).

Зауважимо, що короткий текст елемента, без сумніву, швидко читаючись, абсолютно *необов'язково швидко розпізнається*. Тому не варто дуже скорочувати текст елемента: *викидати потрібно все зайве, але не більше* [1].

Піктограми в меню. Піктограми в меню, *якщо вони повторюють піктограми на панелі інструментів*, мають реальну *здатність навчати користувачів можливостям панелі* [1].

Найважливішою властивістю елемента меню є його *назва*.

Назва має бути найефективнішою з можливих варіантів. Елементи головного меню практично ніколи не несуть на собі контексту дій користувача тому, що в будь-який момент часу доступні всі елементи. Це означає, що до найменування елементів меню потрібно підходити вельми ретельно, ретельніше, ніж до всього іншого.

Особливо варто зупинитися на *відміні* тексту. Єдиним способом розмежування елементів є текст, так що потрібно дуже ретельно підходити до того, щоб елементи, що запускають дії, були *дієсловами у формі інфінітива*.

Втім, часто дієслово доводиться викидати взагалі, щоб перемістити значуще слово ближче в початок тексту елемента. Потрібно це, щоб підвищити швидкість розпізнавання.

Підвищити її можна всього одним способом: *головне слово в елементі повинне стояти першим*.

Тепер менш очевидний: *піктограми краще працюють, коли ними забезпечені не всі елементи*.

Коли всі елементи мають піктограми, розбірливість кожного окремого елемента падає: *врешті-решт, піктограми всіх непотрібних зараз елементів є візуальним шумом*.

Коли ж піктограмами забезпечені тільки *найважливіші елементи*, їх розбірливість підвищується (а розбірливість останніх не знижується), водночас *користувачам вдається легше запам'ятовувати* координати елементів («елемент відразу під другою піктограмою»).

Слід зауважити, якщо головне слово стоїть першим, ця обставина дуже *прискорює пошук відомого елемента* і точність його вибору, так само, як і загальну розбірливість меню. Це очевидний факт.

Елементи, що перемикаються. На особливу увагу заслуговують випадки, коли меню перемикає які-небудь взаємовиключні параметри, наприклад, показувати або не показувати палітру. Тут є декілька можливих способів. Можна помістити перед перемикачем помітку (галочку), показуючи, що він увімкнений (якщо ж елемент забезпечений піктограмою, можна її «втоплювати») [1].

Можна не поміщати *помітку*, зате *інвертувати текст* елемента: наприклад, елемент Показувати сітку перетворюється на Не показувати сітку. Це не ефективно з багатьох причин [1].

По-перше, в інтерфейсі бажано *не вживати нічого негативного*: меншою мірою тому, що *негативність знижує суб'єктивне задоволення*; більшою мірою тому, що вона *знижує швидкість розпізнавання тексту* (головне слово не перше, потрібно зробити роботу, щоб із заперечення обчислити твердження).

По-друге, якщо вилучити «не» і переформулювати один із станів елемента, користувачам буде важче усвідомити, що два різних елементи насправді є один елемент. *Таким чином, позначка більш доречна*.

Передбачуваність дії. Користувачів потрібно забезпечувати відчуттям контролю над системою. Стосовно меню це означає, що за виглядом елемента користувачі повинні здогадуватися, що відбудеться після вибору. Зробити це неймовірно важко, оскільки на екрані немає місця під такі підказки. Можна зробити тільки одне, але зробити це потрібно обов'язково: *потрібно показати користувачам, який елемент запускає дію або змінює параметр, а який відкриває вікно з продовженням діалогу* [1].

Майже у всіх ОС стандартним індикатором продовження діалогу є *три крапки* після тексту елемента.

Так що користуватися цією ознакою необхідно скрізь, включно з інтернетом. Також необхідно показувати. який елемент спрацьовує відразу, а який відкриває елементи меню нижнього рівня (у будь-якій ОС це робиться автоматично, в інтернеті потрібно не забувати робити це вручну).

Це ж правило стосується і гіпертекстових посилань узагалі (вони є також меню). Користувачі отримують значно більше відчуття контролю, коли мають можливість передбачити, куди їх посилання приведе (водночас знижується кількість помилкових переходів).

Таким чином, нестандартні посилання (тобто посилання на інший сайт, на поштову адресу, на файл, на вузол FTP, на сторінку, що довго завантажується тощо) корисно забезпечувати характерними для них ознаками, наприклад, посилання на поштову адресу піктограмою листа.

5.3.2. Метод групування елементів

Під *групуванням* елементів розуміють об’єднання елементів, як правило, за *певною спільною функціональною властивістю*.

У більшості меню групування надається не менше значення при пошуці потрібного елемента, ніж сама назва елемента, бо навіть ідеальна назва не спрацює, якщо елемент просто не можливо знайти (рис. 5.4).

Навіщо елементи в меню потрібно групувати [1]?

Меню, групи елементів у якому розділені, *сканується значно швидше звичайного*, оскільки в такому меню більше «точок прив'язки» (так само, як і в меню з піктограмами).

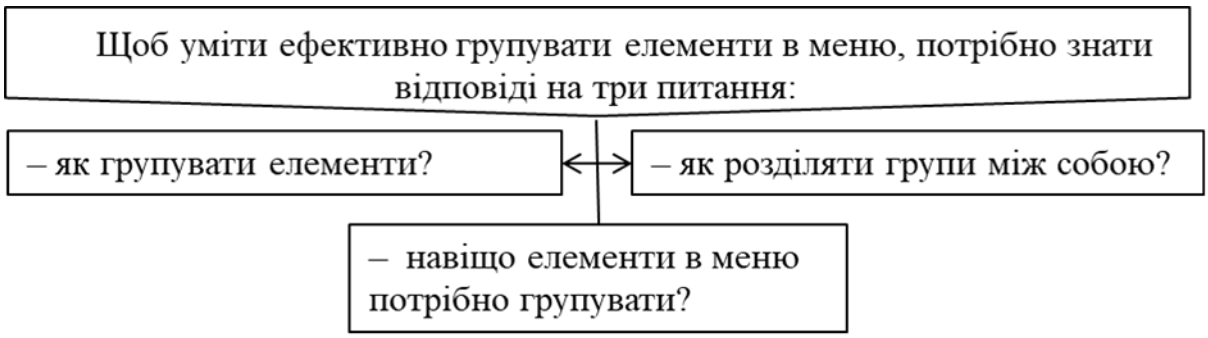

Рис. 5.4. Алгоритм ефективного угрупування елементів

До того ж наявність явних роздільників у *багато разів полегшує побудову ментальної моделі*, оскільки не доводиться гадати, як зв'язані між собою елементи.

Як групувати елементи [1]?

Елементи в меню потрібно групувати *максимально логічно*. Це очевидно, але від цього його проблематичність не зменшується. Річ у тому, що є безліч типів логіки. Є *логіка розробника*, який знає всі функції системи. Є *логіка користувача*, який знає тільки меншу частину. За цих умов практика показує, що *ці типи логіки значною мірою не збігаються*.

Оскільки *користувачі важливіші*, потрібно *згрупувати меню відповідно до їхньої логіки*. Для цього використовується дуже простий і надійний метод, що має назву *карткове сортування*: всі поняття, які потрібно групувати, пишуться на паперових картках з розрахунку «одне поняття – одна картка» [1].

Після чого групі користувачів з цільової аудиторії пропонується ці картки розсортувати (водночас кожен об'єкт отримує свій набір карток). Купки з карток, що вийшли, потрібно розібрати на складові і звести результати від різних суб'єктів в один спосіб групування [1].

Взаємовиключні елементи бажано поміщати в окремий рівень ієрархії

Як розділяти групи між собою [1]? І

Є *два основні способи розділяти* групи:

– між групами можна поміщати *порожній елемент* (роздільник);

– або ж розміщувати окремі групи в *різних рівнях ієрархії*.

Другий спосіб створює чіткіше розділення: у меню Файл, наприклад, усі елементи ближчі один одному (незважаючи на роздільники), ніж елементи інших меню.

У той же час вибір конкретного порядку диктується результатами карткового сортування, так що інтерес становить тільки питання «який вигляд повинні мати і як діяти роздільники».

Для *розмежування груп* традиційно використовують *смужки*. Це надійне, просте рішення. Інша справа, що з дизайнерської точки зору смужки погані, оскільки є візуальним шумом [1].

Набагато правильніше, але і важче, використовувати лише візуальні паузи між групами, як це зроблено, наприклад, у MACOS X.

5.3.3. Метод визначення глибини меню

Вибрана *спільна функціональна особливість* угрупування визначає рівень (***рід***). Оскільки в одну групу входить достатньо *велика кількість* елементів, що *значно ускладнює пошук*, то виникає практична *необхідність звуження* основи. Це досягається *уточненням функціональної особливості* і як результат отримується *новий рівень* (***вид***).

Між рівнями створюються *каскадні* стосунки. Глибина меню визначається можливістю внесення елемента в рівень без порушення каскадності меню [1].

Оскільки *каскадність угрупувань визначає вкладеність рівнів*, то наявність багатьох рівнів вкладеності в меню призводить до так званих «каскадних помилок»: вибір неправильного елемента верхнього рівня неминуче призводить до того, що всі наступні елементи також вибираються неправильно.

Згідно з психофізичними особливостями сприйняття інтерфейсу *для користувача більш зручним є широке меню, у якому візуально відображається найбільша кількість функцій*. Тому найчастіше розробники інтерфейсів прагнуть створювати широкі, а не глибокі меню [1].

При переміщенні по меню користувач діє за алгоритмом:

1) вибираючи елемент першого рівня, він вибирає елемент, «потрібність» якого здається йому *максимальною*;

2) після вибору він бачить список елементів другого рівня, водночас він оцінює вірогідність відповідності всіх елементів другого рівня його завданню й одночасно вибирає найбільш вірогідний елемент. Водночас у думці він тримає контекст, тобто назву елемента першого рівня;

3) якщо жоден з елементів не здається користувачеві достатньо вірогідним, користувач повертається на перший рівень;

4) якщо якийсь елемент задовольняє користувача, він вибирає його й отримує список елементів третього рівня. Дії з другого і третього кроків повторюються стосовно нових елементів меню.

На жаль, *у широких меню є недолік: вони займають багато місця*. Це означає, що, починаючи з певної кількості елементів, меню фізично не зможе залишатися широким, воно почне рости в глибину. Виникає проблема, яку треба вирішувати. Проблема полягає в тому, що *велика вірогідність каскадних помилок*. Щоб знизити їх кількість, *потрібно підвищити вірогідність* того, що користувачі правильно *вибиратимуть елементи верхніх рівнів*. Для цього потрібно заздалегідь забезпечити користувачів контекстом дій [1].

Отже, *дії користувача* при пошуці потрібного елемента *мають циклічний характер*, водночас на кожному кроці є вірогідність помилок. З кожним новим рівнем меню обсягу контексту, який доводиться тримати в голові, безперервно зростає.

Є інший метод, і цей метод є, мабуть, краще, що дав інтернет науці при проектуванні інтерфейсів: як анотацію до елемента можна показувати найбільш популярні елементи наступного рівня.

У цьому випадку користувач може сформувати контекст елемента, не переміщаючись усередину цього елемента, за цих умов вірогідність помилкового переходу значно знижується. Окрім зменшення кількості помилок, така система дає змогу прискорити доступ до найбільш суттєвого пошуку елементів другого і подальших рівнів [1].

До меню, що розкриваються, діє ще один обмежувач глибини. Меню, що розкриваються, досить важкі у використанні, оскільки вимагають від користувачів достатньо тонкої моторики.

Тому головне меню з рівнем вкладеності елементів, більшим ніж три, просто неможливе.

У цілому ширина і глибина меню є, мабуть, найменш значущими чинниками. Набагато важливіше доцільне угрупування, водночас як угрупування, так і структуру дерева меню, все одно краще визначати картковим сортуванням.

5.4. Метод формування контекстного меню

Контекстні меню (англ. context menu) – це меню, що відкриваються після натискання *правої кнопки миші*. Називаються вони контекстними тому, що їх вміст залежить від контексту, інакше кажучи, від того, у якій програмі перебуває користувач і на якому об'єкті він його активував [1]. На рис. 5.5 наведено варіант контекстного меню.

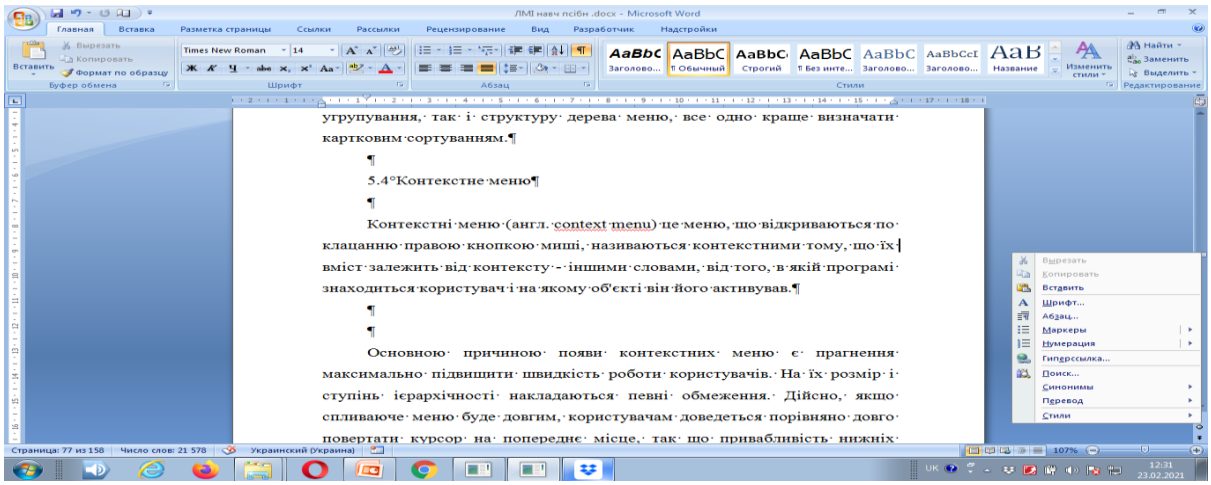

Рис. 5.5. Контекстне меню

Тому краще скорочувати розмір контекстних меню до розумного мінімуму (близько семи елементів).

До того ж не треба забувати, що *головне меню не завжди перекриває виділений* (тобто актуальний об'єкт), а *контекстне меню майже завжди* (оскільки воно викликається на самому об'єкті). У більшості ж випадків перекриття актуального об'єкта небажане (втрачається контекст).

Основною причиною появи контекстних меню є *прагнення максимально підвищити швидкість* роботи користувачів.

На їх розмір і ступінь ієрархічності накладаються певні обмеження.

Справді, якщо *спливаюче меню буде довгим*, користувачам доведеться порівняно довго повертати курсор на попереднє місце, так що привабливість нижніх елементів опиниться під питанням.

Зробити в цій ситуації нічого не можна, окрім того, як зменшити розмір меню у розрахунку, що невелике меню перекриватиме малу кількість інформації.

Інша особливість контекстних меню – ієрархія. У звичайному меню *ієрархія має одну перевагу: при навчанні вона дає змогу упорядковувати елементи меню* і тим самим зробити його зрозумілішим.

У *контекстних же меню навчальна функція не має ніякого значення*, оскільки такими меню користуються лише досвідчені користувачі.

Ієрархія елементів втрачає свою єдину перевагу, не втрачаючи жодного недоліку.

Тому робити ієрархічні контекстні меню можна, але необхідно усвідомлювати, що вкладеними елементами майже ніхто не користуватиметься (вкладеність збиває контекст дій).

Остання відмінність контекстних меню від звичайних полягає в тому, що в них порядок проходження елементів дуже важливий.

У головному меню не обов'язково прагнути до того, щоб найбільш часто використовувані елементи були найпершими – все одно курсор доведеться повертати до робочого об'єкта, так що різниці в дистанції переміщення курсора практично немає [1].

У контекстному ж меню ситуація зворотна: чим далі потрібний елемент від верху меню, тим більше доведеться рухати курсор. Тому правило релевантності в таких меню діє повною мірою [1].

Перевага контекстних (спливаючих) меню полягає в тому, що вони повністю вбудовуються в контекст дій користувачів: не потрібно переводити погляд і курсор в іншу область екрана, практично не потрібно переривати поточну дію для вибору команди.

Водночас вони не займають місця на екрані, що завжди цінно.

Недолік їх у тому, що вони не перебувають весь час на екрані, а тому практично нездатні чому-небудь навчити користувача.

Контрольні запитання до розділу 5

1. Що таке меню?
2. Наведіть типи меню.
3. Охарактеризуйте структуру меню.
4. Поясніть принцип будови окремих елементів меню.
5. Наведіть мету групування елементів.
6. Що означає термін «глибина меню»?
7. Наведіть визначення поняття «контекстне меню».

РОЗДІЛ 6. Проектування елементів управління інтерфейсу

6.1. Кнопки

Кнопкою є елемент управління, уся взаємодія користувача з яким обмежується однією дією – натисненням (рис. 6.1).

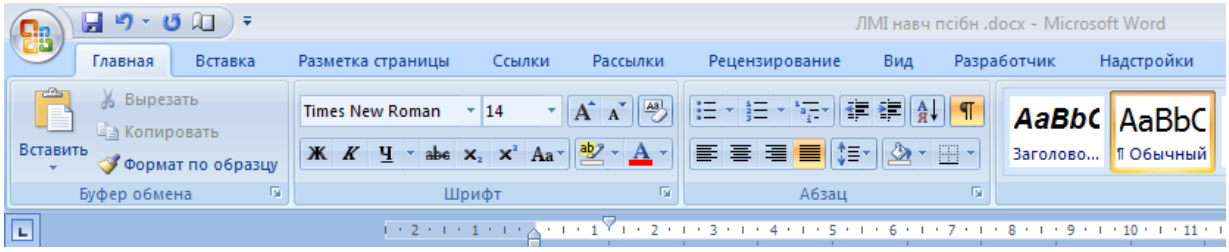

Рис. 6.1. Кнопки як елемент управління

Вважати екранну кнопку за натиснуту потрібно не тоді, коли користувач натискає кнопку миші, а курсор міститься на кнопці, а *тоді, коли користувач відпускає натиснуту кнопку миші*, курсор перебуває на екранній кнопці і перебував на ній, ще коли кнопка миші натискалася [1]. На рис. 6.2. наведено класифікацію клавіш, які входять до складу клавіатури.

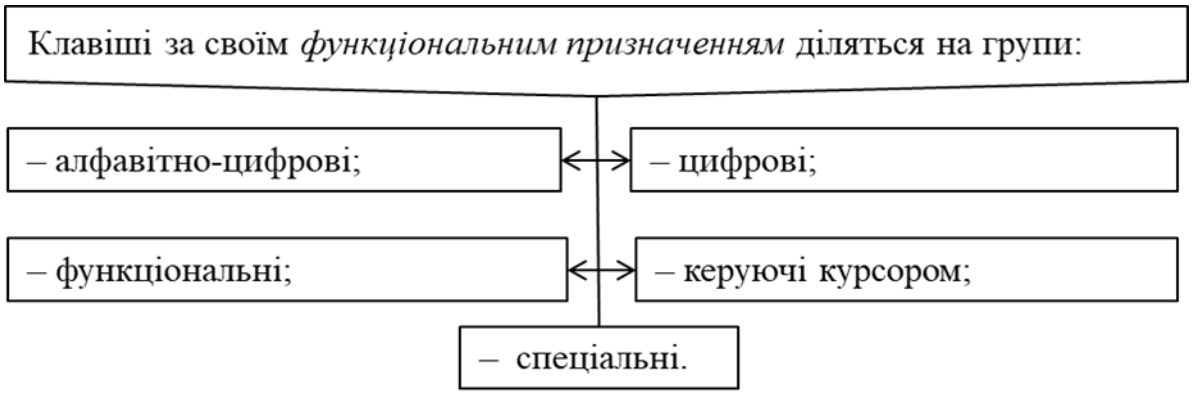

Рис. 6.2. Класифікація клавіш клавіатури

Алфавітно-цифрові призначені для введення текстової і цифрової інформації в комп'ютер. Це *клавіші-символи і клавіші-цифри* центральної частини клавіатури. Умовно до складу алфавітно-цифрових клавіш відносять клавіші: Enter – клавіша введення [1].

Натискування цієї клавіші сприймається комп'ютером, як вказівка *приступити до виконання введеної команди*. Якщо Enter натискується при введенні даних, то це сприймається як вказівка завершити введення даних у рядку і перейти на початок наступного рядка.

Tab – клавіша табуляції. При введенні текстів використовується для фіксації абзацу. Backspace – клавіша витирання символів зліва від курсора.

Цифрові призначені для введення цифр з правої частини клавіатури. Належить до цифрових також: клавіша Num Lock. Якщо її натиснути, то починають працювати клавіші-цифри правої частини клавіатури (загоряється індикаторна лампочка). Повторне натиснення Num Lock блокує роботу цифрових клавіш правої частини клавіатури (індикаторна лампочка гасне) [1].

Функціональні клавіші призначені для виконання певних функцій. Це клавіші від F1 до F12.

Клавіші керування курсором дають змогу переміщатися по всьому екрану. Це клавіші:

– ←, ↑, →, ↓ – зумовлюють переміщення курсора на екрані на одну позицію відповідно;

– Home – переміщення курсора на початок рядка;

– End – переміщення курсора на кінець рядка;

– Pg Up – сторінка вгору;

– Pg Dn – сторінка вниз.

Ці кдавіші використовують у ситуаціях, пов'язаних з *переглядом* на *екрані текстів*, що займають більше, ніж одну сторінку. При натискуванні цих клавіш на екран виводиться попередня/наступна сторінка тексту.

Спеціальні клавіші призначені для змін змісту використання клавіш. Вони виконують певну функцію лише тоді, коли натискаються в парі з іншими клавішами. До складу спеціальних входять такі клавіші:

– Ctrl – клавіша управління;

– Alt – додаткова клавіша;

– Shift – клавіша фіксування великих букв.

6.1.1. Командні кнопки

Натиснення на таку кнопку запускає яку-небудь явну дію, тому їх правильніше називати «Кнопками прямої дії». У той же час цей найпростіший елемент управління має найбільше особливостей, а саме [1]:

– розміри;

– поля.

Чим *більша кнопка*, тим легше потрапити на неї курсором. Це правило зазвичай усіма дотримується, в усякому разі кнопок розміром 5 на 5 пікселів уже практично не зустрінеш.

Проте окрім простоти натиснення на кнопку є інша складова проблеми: *користувачеві повинно бути важко натиснути не на ту кнопку*.

Досягти цього можна або *зміною стану кнопки* при наведенні на неї курсором, або *встановленням проміжку між кнопками*.

Перший спосіб має значну популярність в інтернеті, другий – у звичайному ПЗ. Ні той, ні інший способи не забезпечують стопроцентної надійності. *Отже, за інших рівних умов використовувати варто обидва способи* [1].

Іншою проблемою розміру кнопок в інтернеті є невідповідність видимої площі кнопки та площі, що задіяна [1].

Останнім часом кнопки часто реалізують за допомогою забарвлених комірок таблиці, у яких розміщується текст, що є гіпертекстовим посиланням. Проблема полягає в тому, що користувачі сприймають кнопкою всю комірку, хоча реально «натискається» лише певна мала її частина.

Об'єм. Кнопка повинна бути користувачем натиснута [1]. Відповідно, користувачеві потрібно якось сигналізувати, що натиснена. Кращим способом такої індикації є *додання кнопці псевдооб'єму*, тобто візуальної висоти. З іншого боку, цей спосіб має недолік у тому, що при його використанні виникає диспропорція між видом кнопок прямої і непрямої дії.

Зрозуміло, ніхто не заперечує ще й той факт, що псевдооб'єм кнопок значною мірою є візуальним шумом. Якщо виникає необхідність максимально підвищувати шанси натиснення користувачем якої-небудь окремої кнопки (наприклад, «Про компанію»), у цих випадках псевдооб'єм цієї

кнопки (при інших плоских) дуже підвищує вірогідність натиснення.

Стан кнопки. Кнопка повинна якось інформувати користувачам про свої можливі і поточні стани [1]. Кількість станів досить велика, водночас набори можливих станів у ПЗ і в інтернеті значно розрізняються.

Наприклад, кнопка у Windows може мати *п'ять станів*:
– нейтральний;
– натиснутий;
– нейтральний зі встановленим фокусом введення;
– стан кнопки за умовчанням;
– заблокований стан.

В інтернеті зазвичай використовують менший набір станів: нейтральний, готовий до натиснення (onMouseOver) і активний (у випадках, коли набір кнопок використовується для індикації навігації).

Натиснутий і заблокований стани використовуються дуже рідко, а «нейтральний зі встановленим фокусом введення» створює браузер.

У визначенні станів кнопки головне – відсутність дублювання: не повинно бути різних станів, що мають однаковий вигляд. Також дуже важливо робити заблоковані стани справді заблокованими [1].

Так, наприклад, в інтернеті дуже часто трапляються кнопки, натиснення на які відкриває ту ж саму сторінку, тобто натиснення яких можливе, але марне. Такі кнопки повинні не тільки мати вигляд заблокованих (бути менш яскравими і значними, ніж звичайні), але й не мати гіпертекстових посилань. *Зауважимо, ніколи не видаляйте елементи, які не можна натиснути, замість цього робіть їх заблокованими* [1].

Текст і піктограми. Зазвичай при розробці інтерфейсу *замовник з дивовижною завзятістю вимагає забезпечувати командні кнопки назвами*, поданими дієсловами у формі інфінітива. Розробники ж інтерфейсу з *не менш дивовижною завзятістю не дотримуються цього правила*. Аргументи у кожного свої: у перших – усі так роблять, отже, це є стандарт і

його потрібно виконувати, у других – немає часу придумувати назву [1].

Якщо другий контраргумент особливих пояснень не потребує, то суть першого корисно пояснити. Кнопка, що запускає дію, недаремно називається командною. З її допомогою користувачі віддають системі команди. Команда ж в українській мові формується за допомогою дієслова в наказовій формі. Окрім цього, у дієслівних кнопок є одна велика перевага. Згідно з цими дієсловами зрозуміло, яка дія відбудеться після натиснення [1].

Це дає змогу розмежувати діалогові вікна у свідомості (оскільки різні діалогові вікна отримують різні кнопки). У результаті, через збільшення ступеня унікальності фрагментів системи, навчатися системі виходить краще, ніж з кнопками, однаковими скрізь. Більш того, разом з рядком заголовка вікна дієслівні кнопки створюють контекст, що дуже корисно при поверненні до перерваної роботи.

Таким чином, слід уникати створення кнопок з текстом, що нічого не говорить, оскільки такий текст не повідомляє користувачів, що саме відбудеться після натиснення кнопки.

Водночас є одна особливість. Сучасні інтерфейси заповнені термінаційними кнопками Ок, Відміна (Cancel) і Застосувати (Apply), що, власне кажучи, і дає змогу розробникам посилатися на стандарт. *Однак ці кнопки не є доречним прикладом для повторення.*

6.1.2. Кнопки доступу до меню

Є певні ситуації, коли такі кнопки необхідні і доцільні. Для цього тільки потрібно зробити так, щоб кнопка була одночасно і командною кнопкою, і показувала меню. Для цього потрібно зробити дві речі.

По-перше, потрібно розділити кнопку на дві області, одна з яких запускає дію, а друга – відкриває меню.

По-друге, потрібно організувати такий контекст, при якому результат натиснення на кнопку завжди буде зрозумілим.

Наприклад, це дуже добре працює з кнопками Вперед і Назад. Інший приклад: іноді бувають ситуації, коли дій може виконуватися декілька, але найчастіше потрібна тільки одна. У цьому випадку користувачі дуже швидко навчаються цієї дії,

маючи досить простий доступ до останніх. У такому виконанні кнопки доступу до меню працюють чітко.

Зауважимо, що на області, яка викликає меню, обов'язково має міститися зображення спрямованої вниз стрілки. Ця область повинна розміщатися справа на кнопці, щоб зображення стрілки не заважало сприймати текст або піктограму на кнопці.

6.1.3. Чекбокси і радіокнопки

Радіокнопки і чекбокси є кнопками *відкладеної дії*, тобто їх натиснення не повинне ініціювати яку-небудь негайну дію. З їх допомогою користувачі вводять параметри, які позначаться після того, *коли дія буде запущена іншими елементами управління*. Порушувати це правило небезпечно, оскільки це серйозно порушить ментальну модель користувачів, що склалася. У цьому полягає спільність радіокнопок і чекбоксів (рис. 6.3).

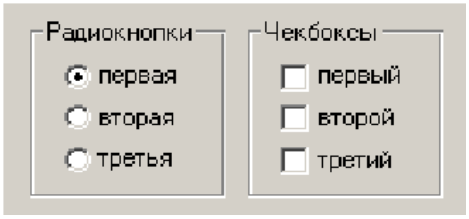

Рис. 6.3. Приклад чекбоксів і радіокнопок

Головна відмінність чекбоксів і радіокнопок полягає в тому, що *група чекбоксів* дає можливість користувачам *вибрати будь-яку комбінацію параметрів*, *радіокнопки* ж дають змогу вибрати тільки *один параметр*.

З цієї відмінності виникають усі інші. У *чекбоксу є три стани* (вибраний, не вибраний, змішаний), а у *радіокнопки тільки два*, оскільки змішаного стану у неї бути просто не може (не можна поєднувати взаємовиключні параметри). У групі *радіокнопок як мінімум одна радіокнопка* має бути *проставлена за умовчанням.*

Зовнішній вигляд. Традиційно склалося так, що *чекбокси мають вигляд квадрата, а радіокнопки – кола*.

Порушувати це правило не можна.

Бажано *вертикально розташовувати чекбокси і радіокнопки в групі*, оскільки це полегшує пошук конкретного елемента.

І *чекбокси, і радіокнопки бажано розставляти по вертикалі*, оскільки це значно прискорює пошук потрібного елемента.

Текст підписів. Кожен підпис повинен однозначно показувати ефект від вибору відповідного елемента. *Оскільки радіокнопки і чекбокси не викликають негайної дії, формулювати підписи до них найкраще у формі іменників,* хоча можливе використання дієслів (якщо змінюється не властивість даних, а запускається яка-небудь дія) [1].

Взаємодія. Це може здатися неймовірним, але і досі в інтернеті 99 % чекбоксів і радіокнопок реалізовано неправильно. Річ у тому, що творці мови HTML вважали, що в цих елементах управління натискається тільки візуальний індикатор перемикання, тобто коло або квадрат. *Насправді це абсолютно не так: натискним має бути ще і підпис, просто тому, що закон Фітса однозначно вимагає великих кнопок* [1].

Але в інтернеті всього цього немає, оскільки в HTML конструкціях чекбоксів і радіокнопок просто не дозволяє робити «нажимабельними» підписи.

Зараз це стало технічно можливим (через тег Label), але за інерцією чекбокси створюють за старою методикою.

Зауважимо, що за необхідності заблокувати елемент бажано візуально послаблювати не тільки квадрат або коло, але і підпис.

Як чекбокси, так і радіокнопки, бувають двох видів: описані вище стандартні, і призначені для розміщення на панелях інструментів (рис. 6.4).

Рис. 6.4. Приклад чекбоксів і радіокнопок на панелі інструментів

На рис. 6.4 зліва розташовано чекбокси (шрифт може бути і жирним і курсивним), справа – радіокнопки (абзац може бути вирівняний або по лівому, або по правому краю) [1].

Зауважимо, що візуально чекбокси і радіокнопки не розрізняються. У них є певний недолік: *вони не розрізняються зовні* (наскільки відомо, ні в одній ОС метод візуального розрізнення не розроблено). Це не дуже важливо, оскільки

панелями інструментів користуються переважно порівняно досвідчені користувачі [1].

Проте на панелях інструментів корисно розташовувати групи радіокнопок *окремо* від груп чекбоксів (щоб вони не змішувалися у свідомості користувачів).

Водночас потрібно пам’ятати, що на панелях інструментів чекбокси і радіокнопки *можуть бути кнопками прямої дії*.

6.2. Списки. Види списків

Список (англ. List box) — елемент (віджет) графічного інтерфейсу користувача, який відображається у вигляді списку елементів, *що прокручується*.

Дає змогу користувачеві *вибрати один або декілька елементів зі списку*, як правило, з утриманням клавіші Ctrl або Shift, щоб зробити множинний вибір.

Найбільш часто використовувані списки *функціонально є варіантами чекбоксів і радіокнопок*.

Швидкість доступу до окремих елементів і наочність у них принесені в жертву компактності (вони економлять екранний простір, що актуально, якщо кількість елементів велика) і розширюваності (простота завантаження в списки динамічно змінних елементів робить їх дуже зручними при розробці інтерфейсу, оскільки це дає змогу не показувати користувачеві свідомо непрацюючі елементи) [1].

Ширина. Ширина списку як мінімум має бути достатня для того, щоб користувач міг визначити відмінності між елементами. В ідеалі, звичайно, ширина всіх елементів має бути меншою від ширини списку, але іноді це неможливо. У таких випадках краще зменшувати текст елементів [1].

Піктограми. Уже досить давно в ПЗ немає технічних проблем з *виводом у списках піктограм окремих елементів*.

Проте практично ніхто цього не робить. Це *погано*, адже піктограми забезпечують *істотне підвищення суб'єктивної привабливості інтерфейсу* і скануються швидше за простий текст.

6.2.1. Списки, що розкриваються

Найпростішим варіантом списку є список, що розкривається. Окрім описаних вище переваг списків, спискам, що розкриваються, притаманна одна істотна перевага.

Вона полягає в тому, що *мала висота списку дає змогу з великою легкістю візуально відображати команди, що побудовані зі складових* (рис. 6.5).

Рис. 6.5. Приклад візуальної збірки команди зі складових

Цей метод значно простіший для розуміння, ніж, наприклад, введення позитивного значення для зсуву вгору і негативного значення для зсуву вниз без підтримки списком, що розкривається.

Список, що розкривається, зумовлює дві проблеми, одна з'являється переважно в ПЗ, друга – в інтернеті.

Перша з них полягає в тому, що іноді *брак місця на екрані не дає змоги використовувати ні чекбокси з радіокнопками*, ні *списки множинного вибору, що перегортаються*. Доводиться робити список, що розкривається, у якому, окрім власних елементів, є «мета-елемент», що містить усі елементи із списку. Такому «мета-елементу» потрібно давати назву, наприклад, Всі значення або Нічого [1].

В інтернеті є інша проблема. Список, що розкривається, часто використовується як навігаційне меню. Це недоречно, оскільки вміст такого меню не видно відразу і вже тим більше його важко відображати користувачам, у якому розділі сайту вони перебувають. Проблема полягає в тому, що список забезпечують скриптом, який запускається відразу за вибором значення [1].

6.2.2. Списки, що перегортаються

Іншим, складнішим варіантом списку є *список, що перегортається*. Списки, що перегортаються, можуть давати

змогу користувачам здійснювати як єдиний, так і множинний вибір [1].

Розмір. По-вертикалі в список повинно поміщатися як мінімум чотири рядки, а краще – вісім. Список по висоті більший, ніж висота елементів, які в нього входять, і якщо він містить порожнє місце в кінці, то має неохайний вигляд.

Списки єдиного вибору. Список єдиного вибору є проміжним варіантом між групою радіокнопок і списком, що розкривається (рис. 6.6) [1].

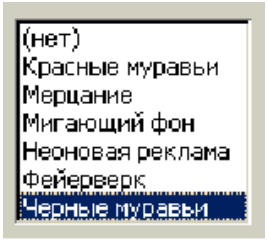

Рис. 6.6. Список єдиного вибору

Він менший від групи радіокнопок з аналогічною кількістю елементів, але більший від списку, що розкривається. Відповідно використовувати його варто тільки в умовах «ледачої економії» простору екрана.

Списки множинного вибору. З погляду дизайну інтерфейсів, списки множинного вибору цікаві перш за все тим, що їх фактично немає в інтернеті. Технічно створити список множинного вибору непроблематично, для цього в HTML є спеціальний тег. Проблема в тому, що такий список у браузері матиме вигляд списку єдиного вибору [1] (рис. 6.7).

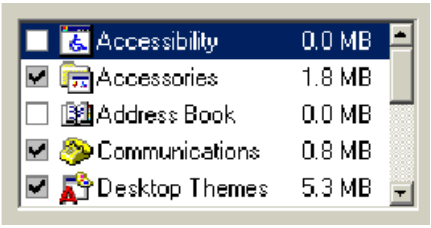

Рис. 6.7. Список множинного вибору з чекбоксами

Більш того, щоб *вибрати декілька елементів*, користувачеві *доведеться утримувати клавішу* Ctrl.

Це означає, що скористатися таким списком зможе тільки мала частина аудиторії.

Через таку незручну реалізацію списків браузерами використовувати їх, *як правило, виявляється неможливо*. Доводиться *використовувати чекбокси*.

Набагато краще йдуть справи в ПЗ. Можливість вільно виводити в списку чекбокси дає змогу користувачам без зусиль використувати списки, а розробникам – без зусиль ці списки створювати.

6.3. Поля введення

Поле введення дає змогу вводити інформацію, наприклад, слово для пошуку чи пароль (рис. 6.8). Разом з командними кнопками, чекбоксами і радіокнопками поля введення є основою будь-якого інтерфейсу. Основними характеристиками поля введення є розміри і підписи [1].

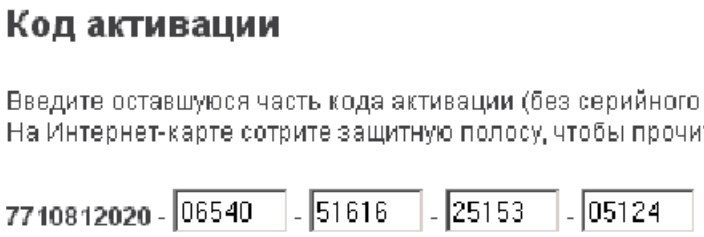

Рис. 6.8. Приклад полів введення великого обсягу

Розміри. Основна частина вимог до полів введення *стосується розміру*. Зрозуміло, що *розмір по вертикалі* має бути похідним від розміру тексту, що вводиться, – якщо тексту багато, потрібно додати декілька рядків. Так *порушення цього правила часто притаманно форумам*, що примушують користувачів вводити повідомлення в поля надзвичайно малих розмірів [1].

Ширина поля повинна відповідати обсягу тексту, що вводиться, оскільки набагато зручніше вводити текст, який бачиш, з іншого боку *ширина поля введення не має бути більшою, ніж обсяг тексту*, що вводиться в поле, оскільки частково заповнене поле як мінімум має неохайний вигляд (рис. 6.6) [1].

6.4. Підписи

Питання «де треба розміщувати підписи до полів введення» є одним з найпопулярніших серед програмістів: битви прихильників різних підходів, хоч і безкровні, але значні. Аргументів і підходів тут багато, але оскільки сприйняття підписів займає певний час, найкраще діє таке просте правило: у часто використовуваних екранах підписи мають бути зверху від поля (щоб їх було легко не читати), у рідко ж використовуваних підписи мають бути зліва (щоб завжди сприйматися і тим самим скорочувати кількість помилок) [1].

6.5. Прокрутки

Прокрутка (spinner, little arrow) є поле введення (рис. 6.9), не таке універсальне, як звичайне, оскільки не дає змогу вводити текстові дані, але має дві корисні можливості, а саме [1]:

– щоб ввести значення в прокрутку, користувачеві не обов'язково кидати мишу і переносити руку на клавіатуру (на відміну від звичайного поля введення). В усякому разі, введення значення в прокрутку з клавіатури достатньо рідкісне, хоч користувачі сприймають прокрутки цілком і повністю позитивно;

– при введенні значення мишею система може дозволити користувачам вводити тільки коректні дані, причому, що особливо важливо, в коректному форматі.

Рис. 6.9. Прокрутка

6.6. Комбобокси

Комбобокс (combo box) – це гібрид списку з полем введення: користувач може вибрати наявний елемент або ввести свій (рис.6.10) [1].

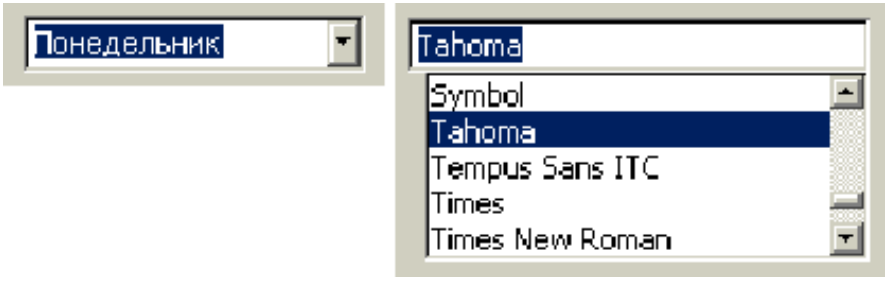

Рис. 6.10. Комбобокс, що розкривається, з встановленим фокусом введення (зліва) і розширений комбобокс (справа)

Комбобокси бувають двох видів: що розкриваються і розширені.

Комбобокси, що розкриваються, мають такий самий вигляд як списки. Візуально відрізняючись від них тільки наявністю індикатора фокуса введення (якщо такий елемент виділений).

Розширені комбобокси, навпаки, можна реалізувати в інтернеті (через JavaScript). Вони мають унікальний вигляд, що відрізняє їх від решти елементів управління. Зате їх порівняно важко (хоча і набагато легше, ніж в інтернеті) реалізувати в ПЗ, оскільки у Windows немає такого елемента, а тому збирати його доводиться з двох. Водночас розширений комбобокс займає багато місця на екрані.

6.7. Повзунки

Як і раніше описані елементи управління, повзунки (рис. 6.11) дають можливість користувачам вибирати значення із списку, не дозволяючи вводити довільне значення. Повзунки незамінні, якщо користувачам треба дати можливість вибору значення, що стоїть у ряду, що добре ранжирується, якщо [1]:

– значень у ряду багато;

– потрібно передати користувачам ранжируваність значень;

– необхідно дати можливість користувачам швидко вибрати значення з великої їх кількості (у таких випадках повзунок виявляється найефективнішим елементом).

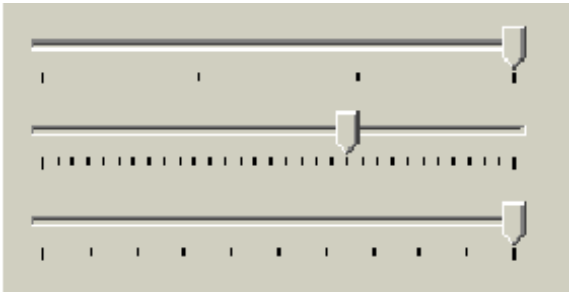

Рис. 6.11. Приклади повзунків

Повзунки мають цікавий аспект. Їх можна також використовувати для вибору текстових параметрів, але лише у випадках, коли ці параметри можна зрозумілим чином відображати.

Контрольні запитання до розділу 6

1. Наведіть визначення поняття «кнопки».
2. Наведіть визначення поняття «командні кнопки».
3. Наведіть визначення поняття «кнопки доступу до меню».
4. Наведіть визначення понять «чекбокси» і «радіокнопки».
5. Наведіть визначення поняття «списки», «види списків».
6. Наведіть визначення поняття «списки, що розкриваються».
7. Наведіть визначення поняття «списки, що перегортуються».
8. Наведіть визначення поняття «поля введення».
9. Наведіть визначення поняття «підписи».
10. Наведіть визначення поняття «прокрутки».
11. Наведіть визначення поняття «комбобокси».
12. Наведіть визначення поняття «повзунки».

РОЗДІЛ 7. Визначення змісту поняття «вікно програми»

7.1. Класифікація типів вікон програм

На рис. 7.1 наведено класифікацію типів вікон програм за їх функціональним призначенням.

Рис. 7.1. Класифікація типів вікон програм за їх функціональним призначенням

Водночас частина окремих типів у загальній сукупності з часом змінюється [1]:

– *вікна документів* відмирають, замінюючись *вікнами програм*;

– *режимні* діалогові вікна заміняються *безрежимними*;

– *безрежимні* у свою чергу – *палітрами*.

Ідея *палітр* теж має тенденцію до зникнення (палітри заміняються *панелями інструментів*).

Так що в майбутньому, найімовірніше, в ПЗ залишаться тільки:

– вікна програм;

– панелі інструментів;

– безрежимні діалогові вікна.

Цікаво те, що порівняно недавно ніяких вікон не було, навіть діалогових вікон, які вже почали сприйматися як даність. Замість них якась частина екрана виділялася під меню, яке в ті часи було функціонально багатшим, ніж меню теперішнє. Так, наприклад, нормою були поля введення в меню.

Зупинимося на структурі вікон різних типів. На рис. 7.2 найбільшим вікном є вікно програми MS Word 2007. Всередині нього розташовується режимне діалогове вікно (Абзац).

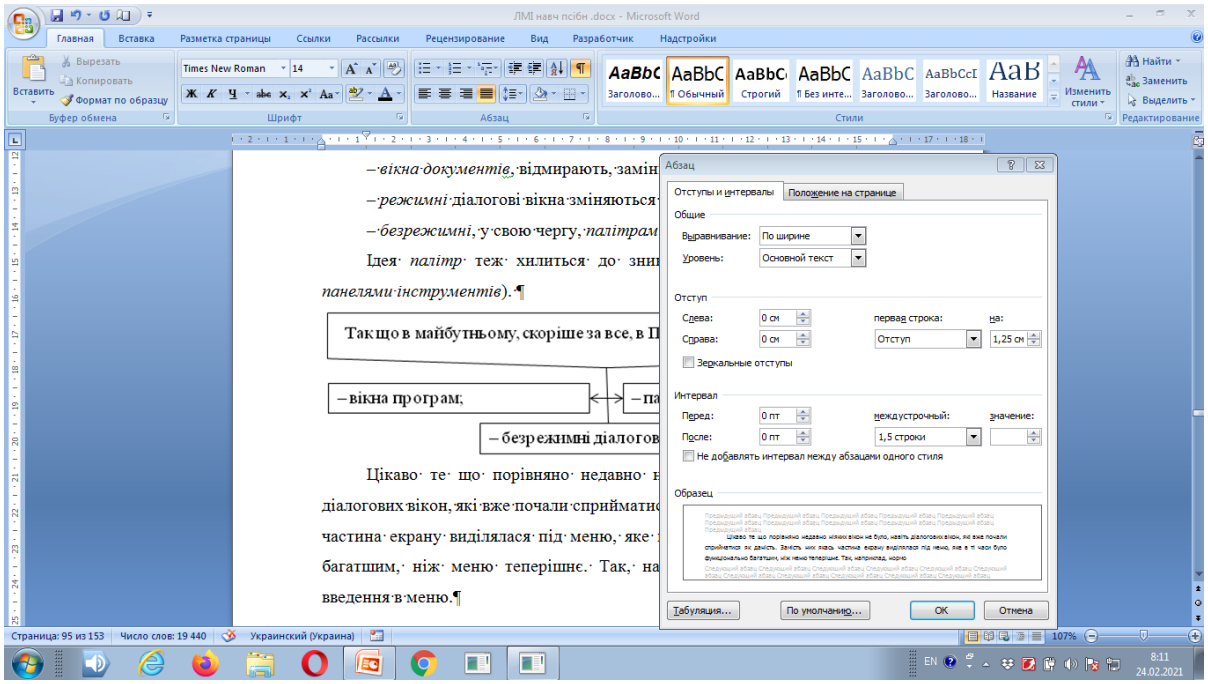

Рис. 7.2. Типи вікон на прикладі MS Word 2007

На рис. 7.3 наведено безрежимне вікно (Знайти і замінити).

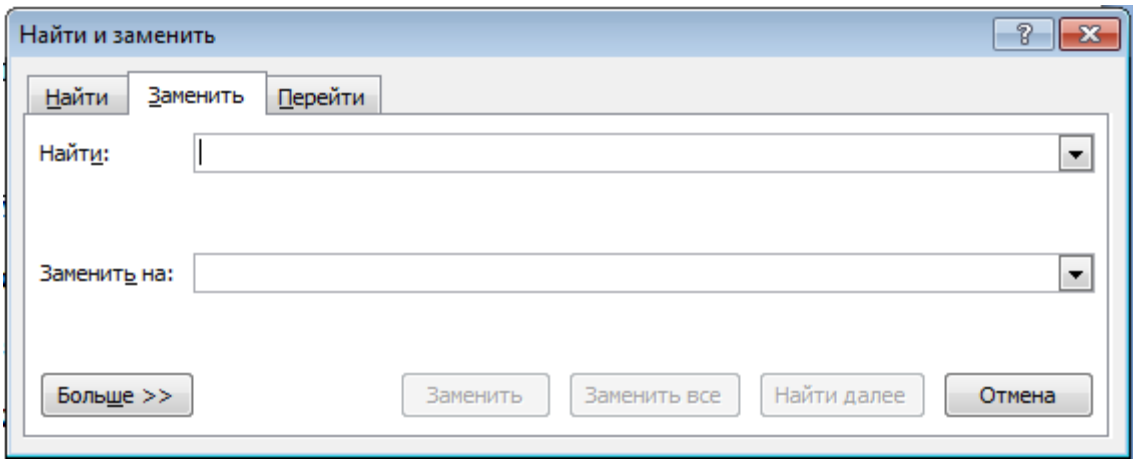

Рис. 7.3. Безрежимне вікно (Знайти і замінити)
MS Word 2007

Зліва внизу на рис. 7.4 для MS Word 97 розташовується палітра (Форматування, отримана відділенням відповідної панелі

інструментів від краю вікна). Зверху і справа дві панелі інструментів (колишні палітри) [1].

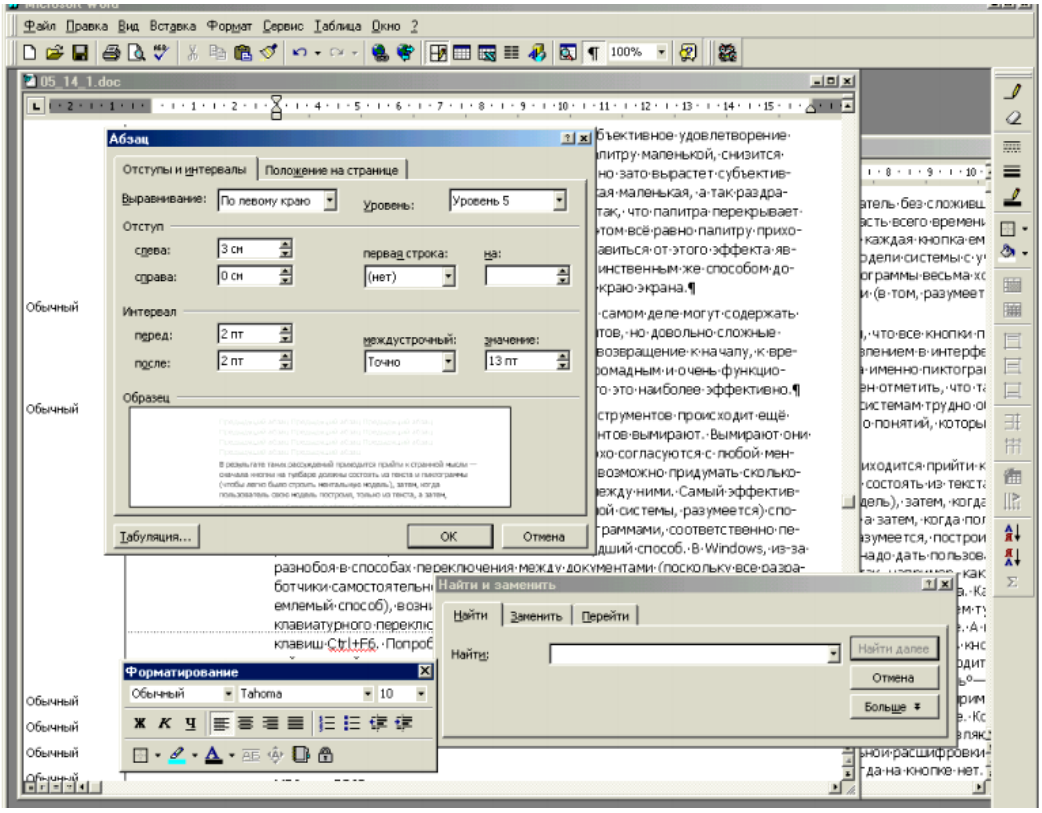

Рис. 7.4. Палітри та панелі інструментів для MS Word 97

З появою графічного режиму стало можливим реалізувати в інтерфейсі метафору робочого столу: з'явилися вікна програм, вікна документів і діалогові вікна, які спочатку були всі режимні [1].

Поняття «*Режимне діалогове вікно*» («Модальне діалогове вікно») здається досить специфічним. Насправді все просто. Якщо вікно, що відкрилося, блокує доступ до іншої частини системи, відбувається, фактично, запуск нового режиму роботи (оскільки функціональність окремого діалогового вікна ніколи не збігається з функціональністю системи в цілому). Після того, як вікно закрите, відбувається повернення попереднього (основного) режиму. У цьому і є все значення терміну «режимний» [1].

З розвитком можливостей інтерфейсу наявність режиму в діалогових вікнах стало недоцільним. А оскільки «дизайн користувачів» був орієнтований на функціонування в реальному світі, вирішили не переробляти користувачів, а переробити інтерфейс.

Так з'явилися *безрежимні діалогові вікна*, тобто вікна, які можна було необмежений час тримати на екрані, перемикаючись у міру потреби між ними і документом. На жаль, і тут не без проблем. Річ у тому, що такі діалогові вікна не можна робити «тонучими», тобто дозволяти користувачеві перекривати їх вікнами документа або програми. Причина проста – користувачі забувають, що вони колись відкривали відповідне вікно і намагаються відкрити його заново [1].

Тому вирішили зробити такі вікна «спливаючими», тобто такими, які перекриваються тільки іншими плаваючими вікнами цієї ж програми або іншими програмами.

Зрозуміло, деякі діалогові вікна неможливо зробити безрежимними: наприклад, повідомленнями про помилки. У цілому з переведенням вікна в безрежимний стан немає особливої проблеми.

Але і тут *виявилася проблема*. Річ у тому, що просто діалогове вікно, навіть будучи безрежимним, малокорисне, оскільки перекриває дуже багато інформації у робочій області [1].

Вирішення цієї проблеми було еволюційним: були придумані *палітри, тобто вікна, з яких вилучили все порожнє місце*.

Виявилось, що *палітри*, окрім малих розмірів, *мають одну велику перевагу*: користувачі дуже люблять їх *розставляти на екрані індивідуальним порядком.*

Це істотно підвищує суб'єктивне відчуття контролю над системою.

На жаль, візуальний дизайн палітр, як правило, досить складний і тривалий, так що суто економічні причини заважають переробити в палітри всі діалогові вікна (рис. 7.5).

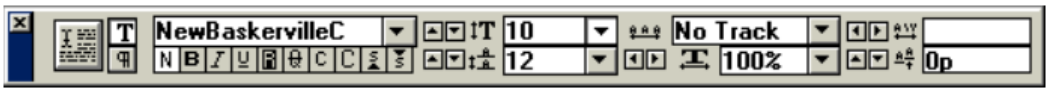

Рис. 7.5. Приклад палітри з програми Adobe PageMaker

На жаль, через малі розміри палітр у них ніколи не поміщаються повноцінні підписи до елементів, що істотно уповільнює швидкість навчання.

Існує неформальний, але на диво правильний закон: *суб'єктивна важливість інформації, що перекривається*

діалоговим вікном (палітрою зокрема), не залежить ні від розмірів, ні від положення вікна, а залежить тільки від периметра.

Водночас, якщо зробити палітру маленькою, знизиться вірогідність її вимушеного перетягування, але виросте суб'єктивне незадоволення від її перетягання («така маленька, а так дратує»).

До того ж палітра перекриває не всю потрібну інформацію, а її частину; при цьому все одно палітру доводиться переміщати. Єдиним способом позбавитися від цього ефекту є *зменшення периметра палітри*, а досягнути цього можна *тільки прикріпивши палітри до краю екрана*.

Так народилися панелі інструментів, які містять не тільки піктограми інструментів, а й не дуже складні елементи управління. В деякому сенсі, це повернення до початку, до часів, коли один з країв екрана був зайнятий величезним і дуже функціональним меню. З іншого боку, виявилось, що це найефективніше [1].

7.2. Визначення змісту головних елементів вікон

Вікна, окрім областей з елементами управління, мають деякі загальні елементи, головними з яких є [1]:

– рядок заголовка вікна

– рядок статусу;

– панелі інструментів;

– смуги прокрутки.

Панелі інструментів та смуги прокрутки були розглянуті вище (тема 6). Зупинимося більш детально на поняттях «рядок заголовку» та «рядок статусу».

Біля кожного вікна є рядок заголовка, але користувачі цим рядком цікавляться дуже мало. Людині не властиво звертати увагу на повсякденність, особливо якщо ця повсякденність не перебуває у фокусі його уваги. З цього, однак, не випливає, що рядком заголовка можна нехтувати. Річ у тому, що текст і меншою мірою, піктограма заголовка мають важливе значення в ПЗ (вони керують перемиканням завдань) і дуже важливе в інтернеті (керують навігацією) (рис. 7.6) [1].

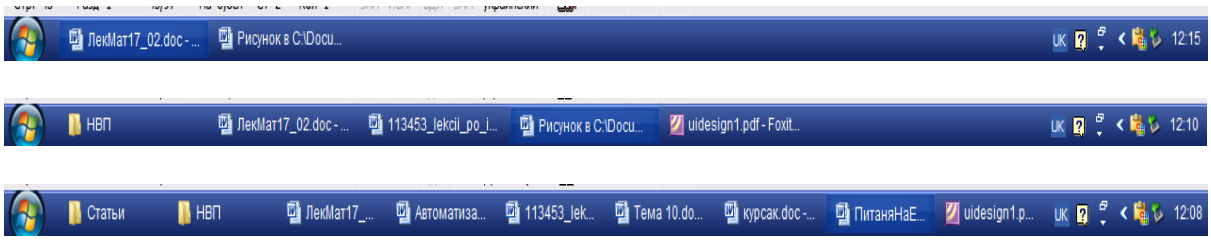

Рис. 7.6. Вплив рядка заголовка вікна на перемикання між завданнями

Натиснення на піктограму в рядку заголовка викликає меню, що розкривається, є зручним місцем для виклику функцій.

Правило релевантності діє і тут – на початку рядка має бути більш релевантна інформація, ніж у його кінці. Оскільки зв'язку «програма-документ» в інтернеті немає, найефективніше показувати адресу поточної сторінки в навігаційній системі сайту (якщо сайт ієрархічний) [1].

Отже, важливо розуміти те, що нехтування користувачем *заголовка вікна*, зовсім не означає, що йому заголовок не потрібний. Навпаки, змістовний заголовок може полегшити розуміння роботи діалогу. *Тому наявність на екрані помітного й адекватного заголовка вікна часто виявляється дуже корисною*. На жаль, у звичайному Windows-интерфейсі місця під нього немає.

Рядок статусу є, мабуть, *найнедооціненим елементом інтерфейсу*. Поширена думка, ніби рядок статусу призначений для того, щоб інформувати користувачів про значення тих або інших елементів інтерфейсу [1].

Насправді цей рядок не може цього робити взагалі: річ у тому, що курсор міститься в одному місці, а підказка з'являється зовсім в іншому, користувачеві при цьому доводиться читати підказку або переводячи погляд, або периферійним зором.

Насправді *рядок статусу* призначений для *двох речей*: він може бути або *власне рядком статусу*, тобто відображати поточний стан системи, або бути *панеллю інструментів* для досвідчених користувачів (або ж робити і те, й інше).

Відображення поточного стану системи. Практично кожна система має властивості, які залежать від документа або змінюються з часом. Наприклад, в ілюстративних програмах

об'єкти мають які-небудь властивості, причому не всі ці властивості показуються [1].

Інший приклад: коли система довгий час зайнята, вона повинна показувати користувачеві індикатор ступеня виконання. І, нарешті, найпростіший приклад: користувач текстового процесора має право знати, на якій сторінці документа він зараз перебуває. Найефективніше виводити все це в *рядку статусу* (рис. 7.7).

Рис. 7.7. Рядок статусу Adobe PhotoShop

На рисунку зліва відображається поточний масштаб відображення документа, далі за ним обсяг пам'яті, що займає документ, потім індикатор ступеня виконання, а справа – контекстна підказка.

Рядок статусу особливо цікавий як місце виведення індикатора ступеня виконання.

Є закономірність: за місцем виведення індикатора виконання можна визначити якість інтерфейсу системи: *якщо індикатор виводиться в рядку статусу, то система має в цілому якісний інтерфейс*, якщо ж індикатор виводиться в іншому місці – неякісний.

Рядок статусу містить у собі деякі важливі елементи з панелі інструментів, які дають змогу досвідченому користувачеві підвищити ефективність роботи системи, а недосвідченому – уникнути помилкових дій, а саме помилкового перемикання (наприклад, кнопки ЗАП, ІСПР, ВДЛ і ЗАМ у статусному рядку MS Word) [1].

7.3. Структура і будова вікна

Структура і сама будова вікна або екрана є найістотнішими чинниками, що впливають на якість інтерфейсу. Наприклад, продуктивність користувачів деколи можна підвищити вдвічі, просто змінивши розташування елементів управління, не змінюючи самі ці елементи [1].

Більшість керівництв з проектування інтерфейсів, перераховуючи вимоги до структури вікна, обмежуються зауваженням, що *термінальні кнопки* (тобто командні кнопки, керівники вікном, наприклад Ok або Закрити) мають бути або знизу вікна, або в правій його частині. Це добре, але мало [1].

Проблеми з'являються у великих вікнах, що дають доступ до багатьох функцій. Зрозуміло, що об’єднувати ці функції в купу неефективно, для цього *інтерфейсні елементи мають бути організовані в блоки*. У ПЗ для цього використовуються переважно рамки угрупування, в інтернеті – пустоти, що розмежовують окремі функції [1].

По-перше, вікно повинне добре скануватися поглядом, тобто його основні частини мають бути відразу помітні. Як правило, у вікнах з малою кількістю елементів управління проблем зі скануванням не виникає.

По-друге, вікно повинне читатися як текст. За інших рівних умов, вікно, у якому всі елементи управління можна без зусиль зв'язано прочитати, краще запам'ятовується і швидше обробляється мозком (оскільки не доведеться перетворювати граматику вікна в граматику мови) (рис. 7.8).

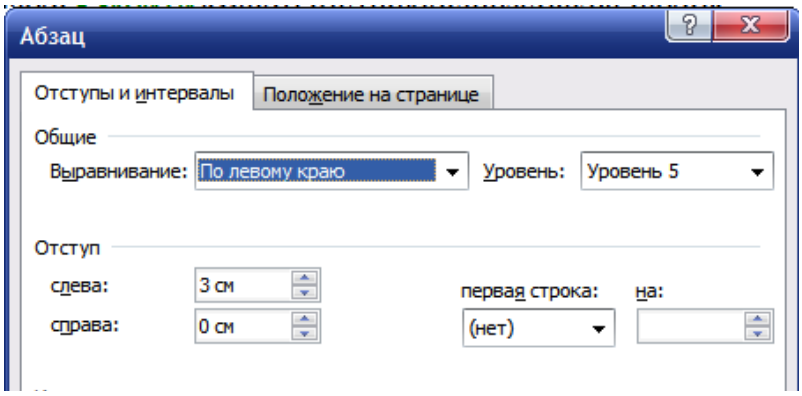

Рис. 7.8. Приклад вікна, яке читається: текст центрується по лівому краю, рівень п'ятий, відступ зліва 3 см, справа 0 см

По-третє, воно повинне задовольняти великий закон «релевантне вперед». Найчастіше використовувані елементи мають бути розташовані в *лівій верхній частині екрана*, рідше використовувані – *в правій нижній частині*.

Зауважимо, що вікно сканується поглядом точно так само, як і відбувається процес читання – *спочатку погляд переходить у лівий верхній кут*, потім *переміщається вправо*, потім переходить на наступний «рядок» і так далі.

Тому, наприклад, вертикальний елемент управління, який розриває ці уявні рядки на частини, завжди уповільнюватиме сканування вікна і викликатиме незадоволення у користувачів.

Термінальні кнопки мають бути розташовані внизу або справа. Річ у тому, що в інтерфейсі завжди повинно бути реалізовано правило: *спочатку вибір параметрів, а потім дія*.

Порушення цього правила істотно підвищує кількість людських помилок і послаблює відчуття контролю користувача (що загрожує низьким суб'єктивним задоволенням).

Це правило, будучи застосованим до діалогових вікон, примушує поміщати термінальні кнопки знизу або справа, тобто в області, яка сканується останньою.

7.3.1. Вкладки

Площа екрана обмежена, а кількість елементів управління, які може знадобитися вмістити в єдиному функціональному блоці (тобто вікні), не обмежено нічим. У цьому випадку доводиться використовувати *вкладки*. Щоб правильно їх використовувати, потрібно дотримуватися певних вимог [1].

Окрім розміщення максимальної кількості елементів управління в діалоговому вікні, *вкладки* можуть виконувати ще одну роль, а саме: *обмежувати інформаційність недосвідчених користувачів* про не дуже потрібну їм функціональність.

Саме тоді, коли потрібно вмістити більше елементів, виникає проблема приховування від користувачів можливо потрібної їм функціональності. Діалогове вікно, що збільшується натисканням кнопки, наведене на рис. 7.9.

Це незручно, оскільки, щоб дізнатися про наявність додаткових критеріїв пошуку, користувачам доводилося здійснювати дуже багато дій. Недивно, що в останніх версіях MACOS ця утиліта була повністю перероблена.

В інтернеті і в решті операційних систем, окрім Microsoft, кнопки, що збільшують розмір вікна і відкривають елементи управління, збереглися в повному обсязі. Ураховуючи той факт,

що жодних користувацьких проблем з ними не відмічено, можна рекомендувати їх і для використання Windows, тим паче, що вони дають змогу досягти певного брендингу [1].

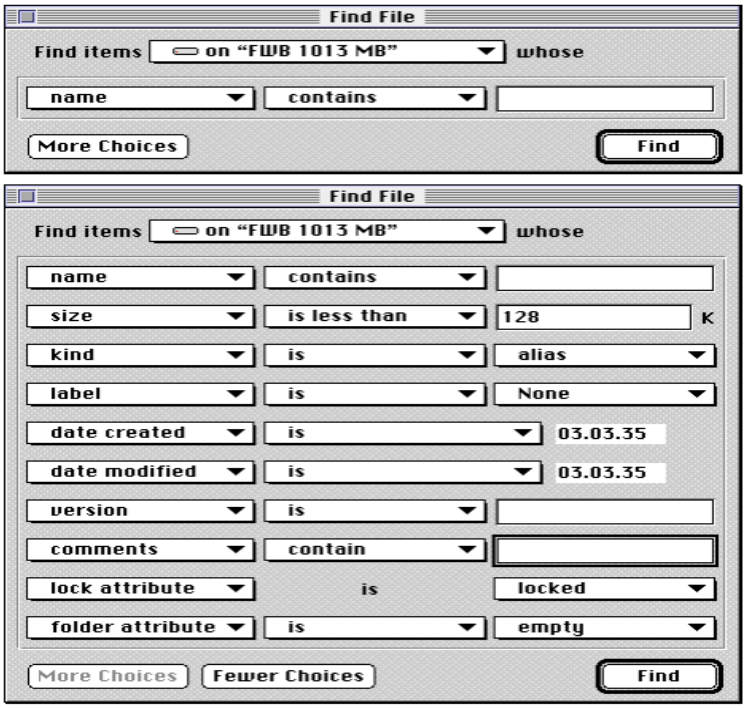

Рис. 7.9. Діалогове вікно, що збільшується. Зверху початковий стан вікна, знизу – кінцевий. Натиснення на кнопку More Choices збільшує вікно

Microsoft усвідомила шкоду від втрати вікон, що збільшуються, та почала використовувати дещо ускладнений елемент управління Tree View, що також дає можливість затиснути багато елементів в обмежений простір. *На жаль, Tree View, якщо вміщати в нього додаткові елементи управління, вкрай незручний у користуванні.*

Теоретично кількість вкладок може бути скільки завгодно великою.

На практиці вона обмежується двома чинниками [1]:

– по-перше. обсягом;

– по-друге, розміром області, у яку *ярлики* вкладок можуть поміщатися.

Річ у тому, що якщо ширина всіх ярликів буде більшою від ширини вікна, доведеться або робити декілька рядків ярликів, або

приховувати частину з них, поки користувач не натисне спеціальну кнопку (рис. 7.10) [1].

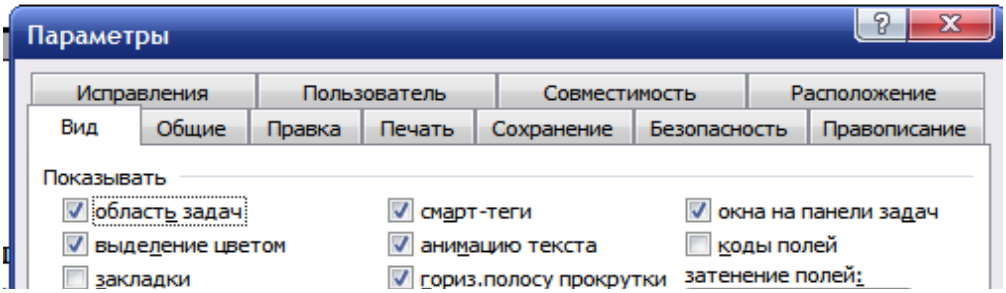

Рис. 7.10. Приклад декількох рядків ярликів

Якщо користувач натискує на ярлику з верхнього ряду (рис. 7.10), весь цей ряд стає нижнім, нижній же переміщається вгору. Водночас два рядки є зовсім не межа – іноді зустрічаються діалогові вікна з більшою кількістю рядів ярликів.

Декілька рядків ярликів погано з двох причин.

По-перше, через велику кількість дрібних деталей (меж ярликів), уся *конструкція досить складно сканується*, тобто важко знайти потрібну вкладку.

По-друге, при послідовному зверненні до декількох вкладок з різних рядів, *ці ряди міняються місцями*, тобто кожного разу потрібно знову шукати потрібну вкладку. Усе це вкрай негативно позначається на суб'єктивному задоволенні і швидкості роботи.

Приховувати частину ярликів також недоречно (рис. 7.11) [1].

Рис. 7.11. Приклад частково прихованих ярликів вкладок

Існує і третій спосіб вирішення проблеми. Можливо просто прибрати вкладки, замінивши їх списком, що розкривається. Цей спосіб теж не дуже результативний, оскільки не візуальний і до того ж порівняно повільний [1].

Очевидно, що найефективнішим рішенням переміщення по ярликах є комбінація другого і третього способів: основні екрани реалізуються у формі вкладок, а додаткові викликаються через

список, що розкривається. Це дає змогу забезпечити максимальну наочність і швидкість роботи.

Фактично кожна вкладка є окремим діалоговим вікном усередині іншого діалогового вікна. Тому дивними є спроби (що трапляються досить часто) *розсортувати елементи управління так, щоб у всіх вкладках їх було порівну*. Робити це у жодному випадку не можна: один екран повинен містити тільки ті елементи, які в ньому потрібні і які користувач може в цьому екрані чекати.

7.3.2. Термінальні кнопки

У діалоговому вікні з вкладками термінальні кнопки обов'язково повинні розташовуватися поза областю вкладок.

Окрім навігації між екранами, є ще й навігація всередині окремих екранів. Користувачам необхідно дати можливість максимально швидко переходити до необхідних елементів управління. Для цього у них є два засоби – миша і клавіатура [1].

З мишею все більш-менш зрозуміло: *оптимізація інтерфейсу за законом Фітса.* Тому оптимізація діалогового вікна, що зменшує дистанцію переміщення курсора, завжди приводить до зростання (хоч і незначного) продуктивності користувачів.

З клавіатурою складніше. Користувач може переміщатися між елементами управління *двома різними способами*: клавішею Tab і «гарячими» клавішами.

Переміщення клавішею Tab здійснюється поволі, зате для цього не потрібно звертатися до пам'яті або шукати клавіатурну комбінацію для потрібного елемента.

7.4. Майстри

Особливим варіантом вікон є *дії, які виконуються за допомогою певної послідовності вікон*, що змінюються одне за одним. Ці дії називаються майстрами (рис. 7.12).

Для розуміння правил щодо них корисно визначити причини, які зумовили появу таких вікон.

Рис. 7.12. Приклад майстра

По-перше, існують дії, для яких або *притаманна, або бажана жорстка послідовність*. Для таких дій єдиний екран, у якому виконується вся послідовність, не дуже ефективний: він загрожує людськими помилками. Тому ефективніше розбити дію на декілька різних екранів [1].

По-друге, існують дії, *які завжди викликатимуть проблеми* у користувачів або тому, *що ці дії складні*, або тому, що використовуються вони рідко (отже, користувачам немає резону вчитися).

Зрозуміло, що *користувачі повинні мати можливість переходити* не тільки *на наступне вікно* в послідовності, але й *на попередні вікна*. Менш очевидним є інша вимога до майстрів: *перехід має бути максимально легким*.

Завдання розкладається на *дві складові*: *перша* – потрібно реалізовувати можливість вільного переміщення по послідовності. Якщо екранів небагато (3–5), то цілком можна обмежитися стандартними кнопками Назад і Далі. Якщо ж екранів багато, перехід по цих кнопках буде, як мінімум, повільним [1].

Незалежно від кількості екранів у послідовності, необхідно *інформувати користувачів про обсяг дії, що залишився* (щоб дати їм можливість оцінити кількість роботи і тим самим підвищувати їх відчуття контролю над системою).

У *довгих послідовностях* показ обсягу екранів, що залишився, може *знизити мотивацію користувачів на початку дії*, але *підвищити мотивацію в кінці*.

Друга складова – чіткість переходу. Для користувачів *майстер*, тобто послідовність екранів, *здається єдиним екраном*, вміст якого змінюється.

Цю ілюзію потрібно підтримувати, оскільки вона дає змогу не збивати направленість дій користувача і підтримувати увагу на «сюжетно-важливій» області екрана.

На відміну від єдиного вікна, у якому виконується дія, у майстрах необхідно підтримувати контекст дій користувача.

Єдиним засобом підтримки контексту є виведення поточного стану даних у процесі виконання майстра.

Майстер чудово підходить до виведення довідкової інформації в самому інтерфейсі.

Довідкову інформацію потрібно виводити *двох типів*, а саме: *короткий опис* і більш розгорнений опис поточного кроку.

З розгорненим описом усе просто. Будь-де внизу екрана (щоб не збивати фокус уваги користувачів) виводиться один або два абзаци, що *надають стандартний набір інформації: що, навіщо і чому*.

З коротким же описом складніше. На жаль, наявне подання майстрів *не має великого і помітного заголовка*. Наприклад, навіть такий прогресивний майстер, як використаний у прикладі (рис. 7.12), не має явно помітного заголовка.

Контрольні запитання до розділу 7

1. Що таке «вікно програми».
2. Назвіть головні елементи вікон.
3. Наведіть структуру і будову вікна.
4. Наведіть структуру і будову вкладки.
5. Наведіть структуру і будову термінальної кнопки.
6. Наведіть структуру і будову майстра.

РОЗДІЛ 8. Норми проектування користувацького інтерфейсу

8.1. Вимоги стандартів до проектування користувацького інтерфейсу

Серед причин невдалої розробки програмних продуктів найчастіше називають дві, які мають прямий стосунок до вимог [2]:

– недолік *ясних, визначених та вимірюваних вимог*;

– *обмежений контроль* за вимогами під час розробки;

– нестача ясних вимог щодо *практичності, КІ, узгодженості, комплектації, користувацьких і ділових потреб та очікувань*.

Найважчу частину збирання вимог становить робота з людьми для визначення їх реальних потреб та запитів [2].

При збиранні вимог потрібно брати до уваги *як технічні, так і соціальні міркування*. На рис. 8.1 наведено класифікацію вимог, які формує колектив розробників.

Рис. 8.1. Класифікацію вимог, які формує колектив розробників до проектування користувацького інтерфейсу

Вимоги до КІ додатків звичайно подані на дуже високому рівні і в них не вистачає специфіки. *Вимоги до КІ залишають великий простір для інтерпретації та уявлення, однак наявні документи можуть служити непоганою відправною точкою для розуміння потреб продукту.*

Вимоги необхідно збирати при розробці будь-якого продукту. *Основна проблема*, яка пов'язана з розроблюваним продуктом (незалежно від етапу розробки), *полягає в збиранні та управлінні введенням нових вимог у процесі розробки*.

Задачі кінцевих користувачів вимагають хорошого опису разом із виясненням розподілу частоти їх виконання. *Крім виявлення задач, які стоять перед користувачем, корисно ознайомитися з процедурами, яких дотримуються користувачі в процесі виконання роботи* [2].

Поряд з описом функціональних можливостей необхідна деталізована *інформація про кінцевих користувачів*, тому що кожен користувач – це особистість з унікальними навичками, поглядами та знаннями.

Використання методів *реверсивної інженерії* дає змогу сформулювати вимоги, *базуючись на наявній системі* (реінжиніринг ПЗ – реконструкція існуючих систем з подальшою переробкою та реалізацією нових проектних рішень).

Перелік функцій, необхідні дані, час відгуку, потік задач, поточна інформація та методи побудови КІ *виводяться* (реконструюються) на *основі аналізу наявної системи*.

Потреби організації та користувачів додаються до вимог та підлягають *перегляду* з погляду їх повноти та пріоритетів.

Поряд з наявними використовуваними продуктами до *інших джерел даних за можливостями КІ*, інформаційною підтримкою та практичністю *належать реальні конкуруючі та аналогічні продукти*. Конкуруючі та аналогічні продукти розглядають з використанням методів, застосовуваних для оцінки наявних продуктів [2].

Є чимало джерел інформації у вигляді галузевих публікацій. Типовий сценарій збирання вимог [2]:

– кінцевий користувач звертається до керівника;

– керівник звертається до представника ІТ- підрозділу;

– представник ІТ-підрозділу звертається до менеджера ІТ-підрозділу;

– менеджер ІТ-підрозділу звертається до спеціаліста з планування продуктів;

– спеціаліст з планування розвитку продуктів звертається до колективу розробників.

На рис. 8.2 наведено склад можливостей користувацького інтерфейсу для формування будь-яких типів інтерфейсів.

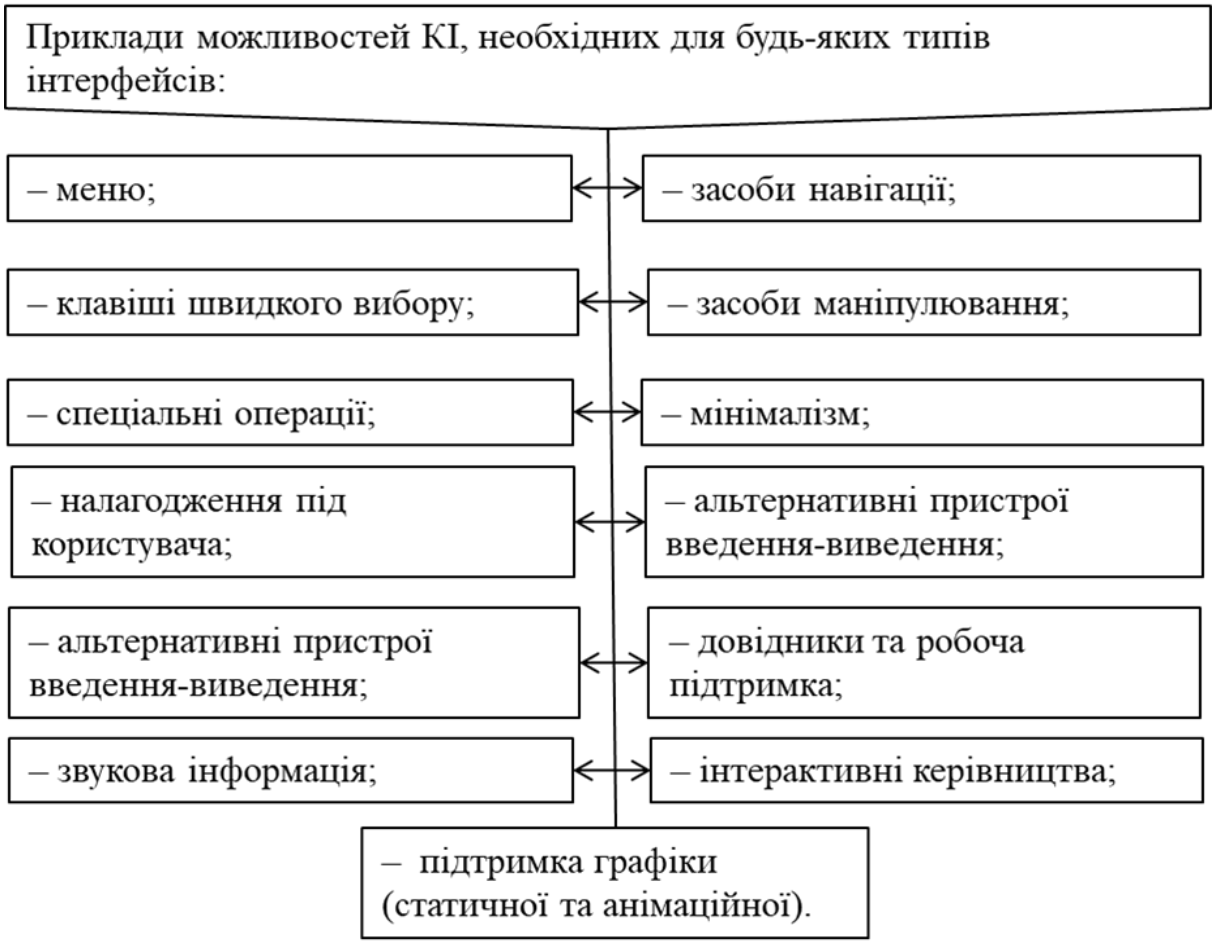

Рис. 8.2. Склад можливостей користувацького інтерфейсу для формування будь-яких типів інтерфейсів

Вимоги найкраще сприймаються у вигляді окремих команд та екранів прототипу КІ, тоді користувачі зможуть оцінити й схвалити властивості та практичність цього прототипу. При збиранні, структуризації та визначенні пріоритетів потреб надзвичайно корисні *таблиці вимог*, оскільки до них можна додавати стовпці для визначення можливостей, які вже наведені в КІ, а також відображення результатів тестування вимог.

На рис. 8.3 наведено склад питань, які повинні враховуватись при формуванні складу вимог до користувацького інтерфейсу.

Рис. 8.3. Склад питань, які повинні враховуватись при формуванні складу вимог до користувацького інтерфейсу

При відстежуванні змін у вимогах необхідна більша дисциплінованість, ніж при початковій фіксації вимог [2].

Якщо функція або характеристика КІ передбачає вимірювання, звітність, відстежування та є основою для дій, то її слід визначати як частину вимог до продукту.

Вимоги щодо практичності легко встановлювати в невизначеному та важкому для вимірювання вигляді. Склад стандартних метрик наведено на рис. 8.4.

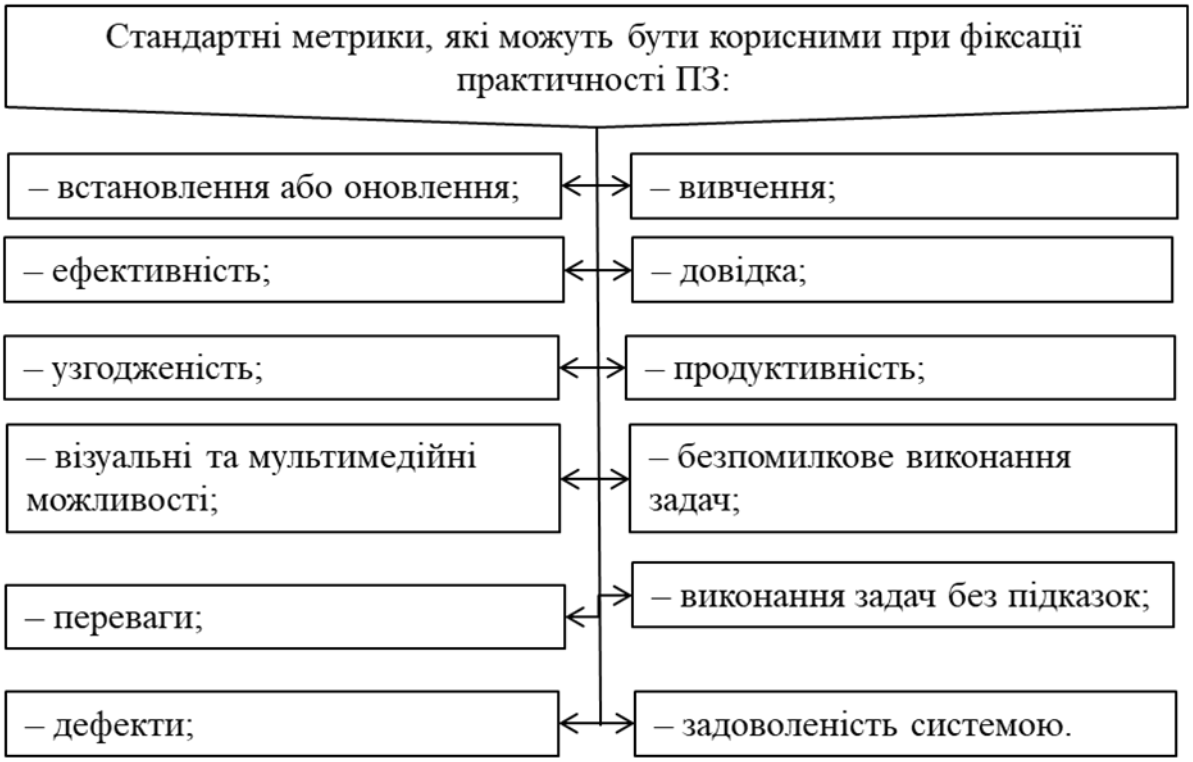

Рис. 8.4. Склад стандартних метрик для фіксації практичності програмного забезпечення

Перелік найбільш важливих характеристик, пов'язаних з виконанням користувачем задач та потоками робіт:

– *кількість* підтримуваних переривань задач;

– *функціональні можливості*, які підтримуються під час переривань та навігації;

– *підтримка КІ* під час переривань.

Стандарти роблять наше життя простішим, розкриваючи характеристики об'єктів і систем, які нас оточують. Стандарти є всюди – це основа індустріалізації. Стандарти комп'ютерного проектування розробляють державні та суспільні організації, інші локальні та міжнародні формації [2].

Найвідоміші організації з розробки стандартів:

– Американський національний інститут стандартів ANSI;

– Німецький інженерний стандарт DIN;

– Міжнародна організація зі стандартизації ISO.

Стандарти є для дисплеїв, клавіатур, системних деталей та ін. Стандарти на ПЗ звичайно застосовні для основних характеристик КІ.

8.2. Керівні принципи та керівництва за стилем

Керівні принципи побудови інтерфейсу розраховані на сьогоднішні системи введення та виведення інформації. Вони починають використовувати такі технології, як використання пера, написання від руки та голосове введення [2].

Одна з проблем розробки інструкцій, які відповідають новим технологіям, – це розшифрування способів взаємодії користувача із системою, оскільки ступінь цієї взаємодії ще точно не визначений.

Інструкції повинні базуватись на тому, як користувачі реагують на новації, та створюватись по проходженні деякого часу, необхідного, щоб користувачі засвоїли інтерфейс та склали певну думку про нього.

Ніколи не встановлюйте стандартів і не пишіть інструкцій, не знаючи, як користувач працює з новітніми технологіями.

У керівництвах з елементів інтерфейсу та його органів управління сказано, коли їх потрібно використовувати, як «подати» і якою повинна бути техніка роботи з ними (клавіатурна чи за допомогою миші). Повний набір керівництв розкриває сутність кожного об'єкта та елемента інтерфейсу в термінах та способах подання на екрані, їх поведінку, механізм взаємодії з ними користувачів [2]. На рис. 8.5 наведено характеристики керівних принципів, які застосовуються при розробці користувацького інтерфейсу.

Більшість програмних продуктів створені для роботи на *різних платформах*.

З тих пір, як ці платформи мають *різні операційні системи*, *інструменти та стилі інтерфейсу*, дуже складно розробляти інтерфейс, який задовольняє *всі платформи* або навіть просто працює на кожній з платформ.

Недавнє доповнення − підбір індустріальних керівництв з проектування − розробив Беллкор.

Задачі керівних принципів проектування однозначні: *надати користувачам можливість доступу до інформації з будь-якого місця системи, у будь-якій формі,* створити такий інтерфейс, який допомагав би людям працювати та подобався їм.

Добре розроблений інтерфейс дає змогу користувачам *сфокусуватись на виконанні задач, а не на особливостях ПЗ та АЗ.*

Рис. 8.5. Характеристики керівних принципів, які застосовуються при розробці користувацького інтерфейсу

Стандарти, промислові керівництва з розробки та стилю оформлення *в порядку спадання пріоритету*:

1) міжнародні стандарти;
2) галузеві керівні принципи на платформах;
3) корпоративне керівництво з оформлення стилю інтерфейсу;
4) керівництво з оформлення стилю групи продуктів;
5) керівництво з оформлення стилю продукту;

Цілі і керівні принципи розробки інтерфейсу повинні бути *реалістичними і доступними* для користувачів. Специфіка того чи іншого бізнесу накладає обмеження на ваше середовище. Ваші керівні принципи з розробки КІ також повинні проходити тестування. Щоб продукт відповідав керівним принципам, необхідно мати підтримку з боку розробників [2].

Сумісність системи та інтерфейсу в деяких випадках *може суперечити очікуванням та побажанням користувачів*.

Не варто дотримуватись керівних принципів в області сумісності лише заради самої сумісності – спочатку потрібно правильно розробити будь-що, а вже потім послідовно застосовувати це до всього інтерфейсу.

Розробка інтерфейсу – більше мистецтво, ніж наука.

Розробка КІ на макрорівні являє собою шаблон КІ – продукт збирається весь цілком та його концепція стає ясна користувачам у міру взаємодії з ним. Як тільки з друку виходять нові керівні принципи з розробки інтерфейсів, продавці інструментарію намагаються якомога швидше включити елементи і технології нових інтерфейсів у запропоновані ним інструменти. Нові інтерфейсні технології завжди є рушійною силою для розробки і проектування все більш досконалих інструментів.

Принципи – це загальні вказівки з проектування, які мають якісний характер та належать до категорії тверджень, яких корисно дотримуватись.

Приклад часто виголошуваного принципу проектування – прагнення до простоти.

Потрібно обрати *не більше 10 важливих принципів*, а потім виробляти інструкції, які містять *вимірювані оцінки для цих принципів*, щоб вони стали *чинними та практичними*.

Інструкція (або керівна вказівка) – *це правило проектування*, корисне при реалізації та легко піддається вимірюванню в термінах відповідності.

Виконання інструкцій легко *піддається безпосередньому вимірюванню, вони об'єктивні і конкретні*. У багатьох керівництвах за стилем *інструкції подаються як правила*, а інші вказівки як *рекомендації.*

Стандарти – це *керівні вказівки*, які відображаються безпосередньо в КІ операційної системи або яких дотримується велика кількість провідних організацій галузі при розробці додатків.

Завжди потрібно дотримуватися *галузевих стандартів,* якщо немає твердої, основаної на думці користувачів, упевненості в доцільності відхилень.

Керівництво за стилем КІ – це *концептуальна та високорівнева специфікація*, яка *описує загальний зовнішній вигляд, поведінку та користувацьку взаємодію*, а унікальні подробиці, які стосуються КІ, опускаються.

Керівництво за стилем слід розробляти, зважаючи на потреби проекту, але водночас *прагнути уникнути незадоволення спонсорів, користувачів проекту та користувачів продукту*.

Керівництво за стилем КІ містить принципи, інструкції та стандарти, застосовувані при проектуванні продукту.

Мета керівництва за стилем полягає саме в тому, щоб допомогти розробникам бути послідовними.

Традиційні керівництва за стилем КІ мають описовий характер, тобто *дається загальна вказівка без конкретизації підказаної директиви*.

Керівництво описового типу слабо конкретизує інструкції або директиви щодо того, як саме відображати об'єкт або дію всередині вікна або бажану поведінку при взаємодії. Розробник повинен виділити конкретні деталі, повністю прочитавши керівництво за стилем, сформувати бажаний стиль, а потім визначити конкретні деталі додатків, які виходять за рамки базисних властивостей КІ [2].

Контрольні запитання до розділу 8

1. Наведіть причини невдалої розробки програмних продуктів.
2. Наведіть вимоги, які формує колектив розробників орієнтований на користувачів.
3. Наведіть принципи розробки ПЗ, які сформулював Берет.
4. Якими повинні бути цілі і керівні принципи розробки інтерфейсу?
5. Наведіть керівні принципи та керівництва за стилем.
6. Охарактеризуйте проблеми проектування міжнародних інтерфейсів.

РОЗДІЛ 9. Методи оцінки практичності інтерфейсу

9.1. Особливості оцінювання практичності інтерфейсу

Основний елемент розробки програмного продукту і КІ – тестування за участю користувачів. Якщо продукт задовольняє вимоги до практичності та інші критерії, він передається клієнтам. Якщо продукт не відповідає критеріям, він допрацьовується в ході чергової ітерації [2].

На рис. 9.1 наведено склад чинників, які визначають практичність та загальний рівень задоволеності користувачів.

Рис. 9.1. Склад чинників, які визначають практичність

Перераховані змінні є базовими чинниками. Збір користувацьких відгуків за кожним з цих напрямків важливий для причинного аналізу проблем та загального оцінювання практичності.

Оцінювання практичності передбачає досягнення цілей, зміст яких наведено на рис. 9.2.

Оцінювання призначено для визначення рівня короткотривалої та довготривалої практичності.

Ретельна підготовка до оцінювання та його виконання сприяють обґрунтованому результату, а вміння вислухати користувачів може забезпечити стійкий та якісний зворотний зв'язок із ними [2].

Незалежно від використовуваного методу при оцінюванні необхідно дотримуватися базової послідовності кроків для залучення користувачів та одержання надійних результатів (рис. 9.3) [2].

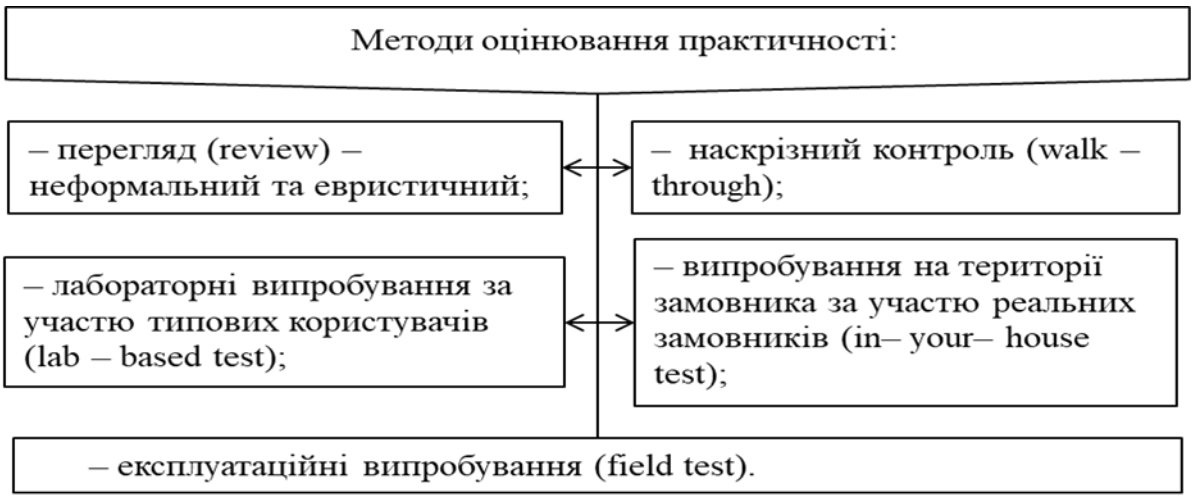

Рис. 9.2. Цілі та методи оцінювання практичності

Рис. 9.3. Базова послідовність кроків для залучення користувачів та одержання надійних результатів

Крім планування та підготовки, оцінювання містить інші основні кроки (рис. 9.4).

Рис. 9.4. Додаткові задачі оцінювання користувацького інтерфейсу

9.2. Визначення змісту поняття «зручності застосування інтерфейсу»

Зручність застосування є «клеєм», який скріплює всі частини, які повинні з'єднатись разом, щоб скласти якийсь продукт. Тобто зручність застосування збирає в єдине ціле такі частини: бізнес-процес; технологію; користувацький інтерфейс; електронну підтримку виконання задачі [2].

На рис. 9.5 наведено перелік категорій за якими оцінюють зручність застосування інтерфейсу.

Вдалий проект інтерфейсу ще не гарантує, що продукт буде зручно використовувати, і в той же час тестування за участю користувачів жодною мірою не заміняє якісної розробки. Обидва питання є частиною процесу розробки інтерфейсу, яка називається *проектуванням зручності застосування* [2].

На рис. 9.6 наведено перелік причин, які обумовлюють важливість тестування користувацького інтерфейсу на зручність.

Міжнародна організація стандартизації (ISO) дає таке визначення: «*Зручність застосування – це ефективність, рентабельність та задоволення, з яким користувачі можуть виконати ті чи інші задачі в заданому середовищі*».

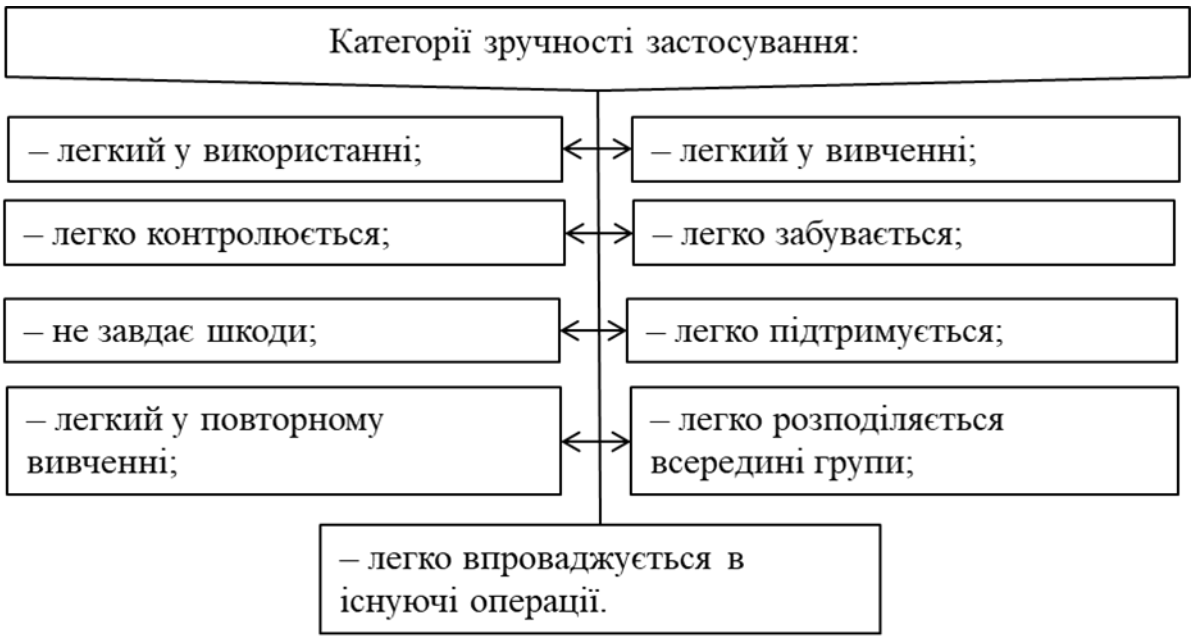

Рис.9.5. Категорії за якими оцінюють зручність застосування інтерфейсу

Рис. 9.6. Причини, які обумовлюють важливість тестування користувацького інтерфейсу на зручність

Тестування на зручність застосування проводиться для того, щоб оцінити якість роботи продукту і з'ясувати, наскільки він ефективний, рентабельний та чи задоволені ним користувачі.

У наш час проектувальники програм орієнтуються на задачі, які стоять перед споживачами. Подібний проблемно-орієнтований підхід повинен виконуватись при розробці КІ, включно із системою електронної допомоги та технічною документацією.

Уже на початковому етапі розробки тестування на зручність застосування дасть змогу нам зрозуміти, чи зможуть споживачі взагалі використовувати цей продукт. Для відповіді на це питання необхідно одержати підтвердження того, що створюваний додаток виконуватиме потрібні користувачам функції та забезпечить їм додаткові переваги в бізнесі [2].

Тестування на зручність застосування здійснюється на різних етапах розробки продукту, щоб забезпечити зворотний зв'язок з користувачами. Це допомагає вдосконалювати весь проект у цілому, скорочує кількість помилок, проводить порівняльний аналіз продуктів і версій, а також підтверджує відповідність продукту вимогам, які до нього висуваються. Тестування є частиною загального процесу розробки зручності застосування.

Способи проведення тестування [2]:

– спостереження; проведення опитів та досліджень; контекстуальні опитування;

– евристичні оцінки; робота з виділеними групами; лабораторне тестування.

Методи оцінювання зручності застосування:

– методи оцінювання роботи, які *передбачають підрахунок дій*, визначення повноти виконання задачі, підрахунок часу, помилок та звернень за допомогою. Такі методи називають числовими;

– суб'єктивні методи, які *містять збір усних та письмових повідомлень* користувачів про їх сприйняття, думки, судження, переваги, а також ступінь задоволеності системою та їхню власну виконану роботу. *Ці методи носять назву якісних*.

9.3. Особливості тестування користувацьких інтерфейсів

Перш ніж планувати і проводити тестування на зручність застосування продукту слід чітко визначити цілі і задачі, які стоять перед ним.

Бут виявив чотири чинники, що становлять зручність застосування [2]:

– корисність;

– ефективність;

– простота вивчення;

– ставлення користувача.

Шекель теж розбиває зручність застосування на чотири схожих категорії [2]:

– простота вивчення;

– ефективність;

– гнучкість;

– ставлення користувача.

В останні роки було проведено ряд порівняльних аналізів, спрямованих на визначення переваг графічних КІ (ГКІ) над традиційними текстовими КІ (ТКІ) для виконання стандартних користувацьких задач. Одним з найбільш відомих досліджень є звіт Wharton Report під назвою «The Value of GUIs», який пропонує сім переваг ГКІ на основі одержаних результатів тестування [3].

На рис. 9.7 наведено переваги графічних користувацьких інтерфейсів над традиційними текстовими користувацькими інтерфейсами для виконання стандартних користувацьких задач.

Рис.9.7. Переваги графічних користувацьких інтерфейсів над традиційними текстовими користувацькими інтерфейсами

Контрольні запитання до розділу 9

1. Охарактеризуйте особливості оцінювання практичності користувацького інтерфейсу.
2. Охарактеризуйте вимогу практичності ПЗ.
3. Наведіть методи оцінювання практичності.
4. Наведіть чинники, які мають найбільше значення для оцінювання.
5. Охарактеризуйте вимогу зручності застосування програмного продукту.
6. Наведіть зміст категорій зручності застосування інтерфейсів.
7. Охарактеризуйте особливості тестування користувацьких інтерфейсів.

РОЗДІЛ 10. Характеристика етапів розробки інтерфейсу

10.1. Принципи розробки інтерфейсу

Розробка, навіть з погляду простої зручності, вимагає команди спеціалістів, які мають таланти в абсолютно різних галузях.

Якщо просто дотримуватися принципів проектування, керівництв і стандартів, то це абсолютно не означає, що буде створено зручний інтерфейс [2].

Немає універсального способу розробки та проектування, який *гарантує* успішний кінцевий продукт. У будь-якому разі потрібно дотримуватись колективного підходу.

Без сумніву, проектування і розробка вимагають навичок у галузі конструювання і створення ПЗ. Крім того, не зайвими в команді будуть графічні та промислові розробники; спеціалісти з психології, які розуміються на пізнавальних та моторних здібностях людини; професіонали, які займаються написанням технічної документації; спеціалісти з тренінгу, які знають проблеми організації праці; а також люди, компетентні щодо пристроїв введення, технологій відображення, інтерактивних методів, діалогового проектування та методології розробки. А оскільки в інтерфейсах усе частіше застосовуються звук, голос, відео, анімація та тривимірна графіка, доводиться залучати й спеціалістів з цих галузей [2].

Ідеальна команда для розробки програми повинна мати такі навички [2]:

- проблемний аналіз;
- програмування;
- розробка КІ;
- розробка команд;
- графічне проектування;
- написання технічної документації;
- тестування на зручність застосування.

У минулому розробка ПЗ та КІ розвивались лише завдяки еволюції технологій та систем, на базі яких програми будувались. Це називалось *системно-керованою або технологічно-керованою розробкою*.

Інтереси користувачів *абсолютно не брали до уваги*. Їм пропонували програмні функції з інтерфейсом, який розробники могли запропонувати. З початку 80-х років акцент було перенесено на розробку, орієнтовану на користувача, причому до розробки залучались і самі користувачі [2]. На рис. 10.1 наведено склад та зміст керуючих принципів, які визначають зорієнтованість розробки інтерфейсу саме на користувача.

Рис. 10.1. Склад та зміст керуючих принципів, які визначають зорієнтованість розробки інтерфейсу на користувача

Проектування КІ може здійснюватись як окремо, так і сумісно з останнім процесом розробки продукту.

Зараз більшу увагу приділяють елементам інтерфейсу та об'єктам, які сприймає і використовує користувач, а не функціональності програм. У багатьох проектах власне розробка

КІ та програмування продукту ведуться паралельно, особливо на ранніх стадіях. На більш пізніх етапах ураховують вимоги КІ та зворотного зв'язку, які виявляються в результаті тестування на зручність застосування.

Процес складається з чотирьох основних етапів [2]:

– збирання та аналіз інформації від користувачів;
– розробка КІ;
– побудова КІ;
– підтвердження якості створеного КІ.

Цей алгоритм може використовуватись як при розробці ООКІ, так і при проектуванні традиційних проблемно-орієнтованих інтерфейсів або ГКІ. Цей процес не залежить від матеріальної та програмної платформ, ОС, застосовуваного інструментарію.

Будь-який вдалий процес розробки КІ повинен бути ітераційним.

Словник Webster New Collegiate Dictionary дає таке визначення слову «*ітераційний*» – «*комп'ютерна процедура, де повторення циклу операцій дає результат, все ближчий до шуканого результату*».

Отже, вдалий інтерфейс неможливо одержати без періодичного повернення до попередніх етапів.

Критерієм для *завершення ітераційної розробки* повинен бути той факт, що *всі вимоги користувачів задоволені*, а сам продукт *відповідає запланованим цілям*.

Може здаватись, що ітераційний процес займає багато часу через численні проходження етапами розробки. Але якщо діяти правильно, то початкові проходження етапами допомагають створити варіанти розробок та прототипів, які в наступних ітераціях зекономлять час на впровадження та тестування.

10.2. Склад етапів розробки інтерфейсу

10.2.1. Перший етап: збір та аналіз інформації від користувачів

Перший етап – дії зі збирання та аналізу інформації – може бути розділений на чотири кроки [2]:

– *визначення профілю користувача* – профіль користувача відповідає на питання «Що являє собою ваш користувач?», він

дає змогу скласти уявлення про вік, освіту, переваги користувача, одержати іншу необхідну інформацію;

– *аналіз задач, які стоять перед користувачем*, – це визначення того, чого хочуть користувачі і яким чином вони збираються вирішувати свої задачі;

– *збирання вимог, наданих користувачами*, – відповідає на питання «Яку, з погляду користувача, користь принесе їм пропонований продукт чи інтерфейс?»;

– *практично в усіх проектах ПЗ враховуються вимоги користувачів*, що допомагає визначити особливості проекту та структуру КІ.

На рис. 10.2 наведено склад спільних вимог користувачів.

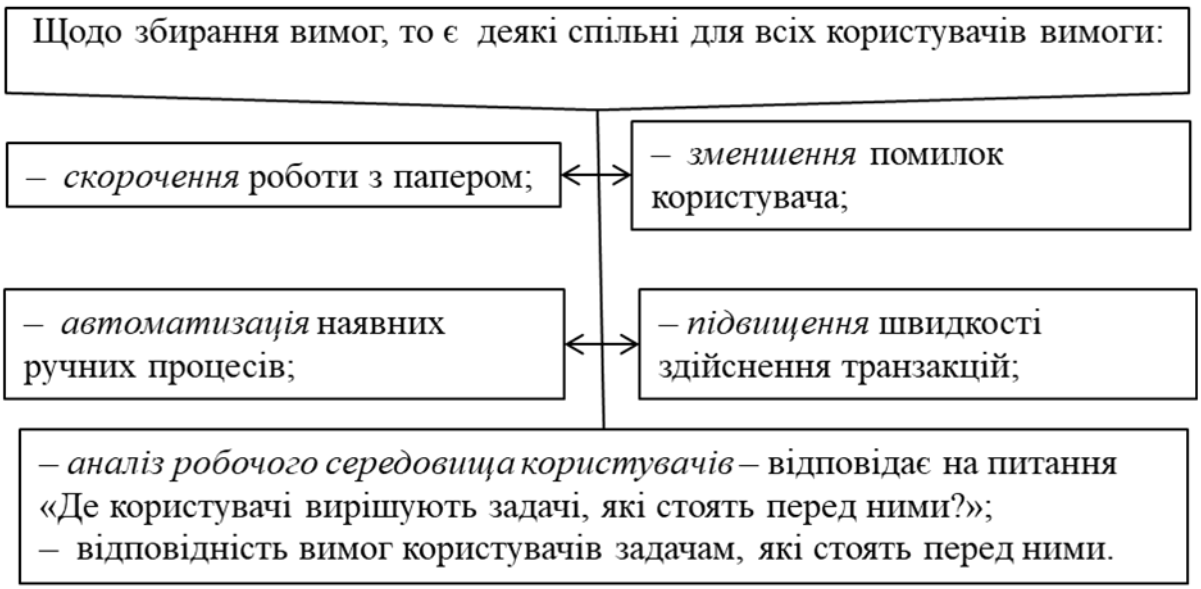

Рис. 10.2. Склад спільних вимог користувачів

Ітераційний метод передбачає повернення до етапу аналізу вимог користувача, щоб перевірити, чи не змінились у процесі проектування і розробки профіль користувачів, задачі, характеристики середовища або самі вимоги.

10.2.2. Другий етап: розробка КІ

Розробка КІ для програмного продукту звичайно вимагає значних витрат часу і ресурсів. Етап розробки складається з певних кроків, виконуваних у заданій послідовності.

Є велика спокуса почати програмування фінальної версії продукту вже зараз, не розробляючи інтерфейс [2].

Однак *потрібно пройти всі етапи процесу розробки*, перш ніж перейти до програмування.

Розробка містить у собі такі кроки:

1) *визначення мети з погляду зручності застосування продукту:*

– на ранніх стадіях розробки продукту потрібно точно визначити, що він зможе зробити для користувачів;

– *цілі розробки* найкраще сформулювати в термінах, які характеризують дії користувача.

Чотири області, найбільш придатні для встановлення цілей і задач:

– *придатність;*

– *ефективність;*

– *легкість у засвоєнні;*

– *оцінка користувачами якості продукту;*

2) *розробка задач і сценарію дії користувачів:*

– *сценарій є описом дій*, виконуваних користувачем, це *послідовність задач*, які стоять перед користувачами, або *подій*, спрямованих на досягнення єдиної мети;

– *слід розробити декілька сценаріїв користувача*, причому чим більше їх буде, тим менша ймовірність, що буде пропущено ключові об'єкти чи операції, необхідні в інтерфейсі;

3) *визначення цілей та операцій інтерфейсу:*

– найскладніший і найважливіший крок;

– на цьому кроці потрібно:

– *виділити об'єкти, дані і дії зі сценаріїв та задач,* які стоять перед користувачами;

– *переглянути й уточнити список об'єктів і дій* сумісно з користувачами;

– *накреслити діаграму взаємодії* між об'єктами;

– *заповнити матрицю* прямого маніпулювання об'єктами;

4) *визначення іконок об'єктів та візуального подання* – визначення, як найкраще *подати визначені об'єкти на екрані*, у якому *вигляді їх побачать користувачі*, яку *інформацію* вони міститимуть.

При визначенні подань об'єктів слід ураховувати *спосіб спілкування користувачів з кожним з об'єктів і з інформацією*, яка міститься в них;

5) *розробка меню об'єктів і вікон* – вияснення, як користувачі будуть спілкуватись з визначеними і розробленими об'єктами і вікнами.

Слід відповісти на такі питання:

– які дії властиві кожному об'єкту і типу;

– що міститься у спливаючих меню;

– яким вікнам потрібна панель меню;

6) оптимізація візуальної розробки.

10.2.3. Третій етап: побудова КІ

При першому зверненні до *ітераційного процесу* розробки потрібно імовірніше *створювати прототип*, ніж будувати КІ [2].

Прототипування є виключно цінним способом створення перших проектів і демонстрації продукту, особливо на ранніх етапах тестування на зручність застосування.

Мета прототипування — швидко і легко візуалізувати різні альтернативні варіанти розробки, а не створювати код, який повинен стати частиною продукту.

Необхідно дотримуватися *трьох «золотих» правил при використанні прототипів* як частини процесу розробки інтерфейсів:

– прототипуйте на ранніх стадіях і не забувайте про ітераційний принцип розробки;

– створюйте різні альтернативні варіанти;

– будьте готові викинути код прототипу.

10.2.4. Четвертий етап: підтвердження якості КІ

Тестування на зручність застосування є ключовим елементом ітеративного процесу розробки. Воно полягає в тому, щоб видати продукт на руки великій кількості користувачів і подивитись, чи зможуть вони з ним працювати.

Мета тестування на зручність застосування повинна полягати в оцінюванні поведінки, дій і ступеня задоволеності користувачів [2].

Більшість розробників звертаються до такого виду тестування *ближче до кінця проектування*. Однак це надто пізно, щоб на основі його результатів вносити зміни. Навіть якщо вони й вносяться, не можна бути впевненими в тому, що виправлений

продукт можна використовувати без проведення повторного тестування.

Розробники повинні обов'язково бути присутніми при проведенні тестування. Тоді вони зможуть побачити, як користувачі працюють з їх продуктами. *Однак вони не повинні мати змоги здійснювати технічну підтримку користувачів під час тестування*.

Контрольні запитання до розділу 10

1 Наведіть зміст етапів розробки користувацького інтерфейсу.

2. Охарактеризуйте ітераційну природу розробки інтерфейсів.

3 Наведіть зміст першого етапу: збір та аналіз інформації від користувачів.

4 Наведіть зміст другого етапу: розробка КІ.

5. Наведіть зміст третього етапу: побудова КІ.

6. Наведіть зміст четвертого етапу: підтвердження якості КІ.

7. Наведіть характеристику ідеальної команди для розробки програми.

8. На що орієнтована розробка інтерфейсу?

9. На яких керуючих принципах основана розробка, яка орієнтована на користувача?

РОЗДІЛ 11. Інструментарій розробника інтерфейсів

11.1. Інструментарій розробника. Передача інформації візуальним способом

На рис. 11.1 наведено перелік інструментів, які використовуються при розробці користувацького інтерфейсу.

Рис. 11.1. Перелік інструментів, які використовуються при розробці користувацького інтерфейсу

Призначення будь-якого інтерфейсу – забезпечити комунікацію між двома системами у цьому разі – людьми та комп'ютерами.

Практичні поради і рекомендації до розробки ПЗ та КІ основані на *традиційних способах, використовуваних при спілкуванні людей*.

Галузі візуального та графічного проектування базуються на знанні теорії мистецтва та візуального сприйняття.

Графічне відображення інформації надзвичайно важливе. Кожен рядок, кожен керуючий елемент, блок, частина тексту, колір і рисунок, які з'являються на екрані, впливають на користувача як окремо, так і в комбінації з усім іншим [2].

З додаванням на екран кожного елемента візуальний вплив на користувача посилюється.

Під час роботи з *візуальним поданням інформації* слід розглядати два чинники:

– *графічна перевага* – складається з комплексних ідей, пов'язаних з ясністю, точністю та ефективністю;

– *графічна цілісність* – являє собою використання графічних засобів для точного відображення даних.

Колір є найменш використовуваним елементом у розробці КІ, хоча проведені дослідження і дані рекомендації щодо застосування кольору як в апаратному, так і в програмному забезпеченні комп'ютерних систем.

Колір в КІ повинен застосовуватись дуже акуратно, адже люди мають різноманітні фізіологічні, психологічні, культурні та емоційні реакції на кольори. *Часто він використовується лише як декоративний елемент*. Це обмежує його здатність відображати значущу інформацію в інтерфейсі. *Один з основних принципів використання кольору – не нашкодити* [2].

У нашій свідомості кольори асоціюються з емоційним станом. Люди мистецтва знають, що *теплі кольори (червоний, оранжевий та жовтий) збуджують і активують*, а *холодні (синій, фіолетовий, бордовий, сірий)* мають *заспокійливу та миротворчу дію*.

Використання кольору в КІ може викликати труднощі у людей, які мають деякі проблеми із зором (кольорова сліпота – дальтонізм). Цей тип зорового дефекту має 8 % чоловічого населення та менше 1 % жіночого населення. Ураховуючи, що користувачами комп'ютерів є люди різного віку та стану здоров'я, розробникам доводиться брати до уваги багато нюансів при створенні апаратного забезпечення монітора та розробці ПЗ екрана [2].

Кольори надзвичайно корисні, оскільки надають інформації додаткового значення. Оскільки колір є сильним засобом фокусування уваги, *велика кількість відтінків змусить користувачів звернути увагу на екран*. Це допомагає зробити інтерфейс більш дружнім та легким у використанні. *Однак цей «ефект Лас-Вегаса» може відвертати увагу користувачів під час роботи*.

ОС пропонують стандартні кольорові схеми і палітри, які й слід використовувати під час розробки інтерфейсу. Вони були створені, тому що кольори складно узгоджувати, і люди часто помиляються при виборі кольору або кольорової комбінації.

Знання теорії сприйняття та особливостей зору спричинило створення дуже важливих фундаментальних рекомендацій до використання кольору незалежно від засобу подання інформації. Ці рекомендації застосовні до використання кольору в друкованих матеріалах, а також на комп'ютерних дисплеях [2].

Робота Мерча широко відома як авторитетне дослідження в галузі психологічного та фізичного аспекту кольору. Мерч пропонує три групи рекомендацій:

– з погляду фізіології;

– з погляду сприйняття;

– з погляду пізнавальності.

Маркус – відомий розробник інтерфейсів і консультант у галузі комп'ютерної індустрії – застосовує такі принципи до використання кольору:

– організація;

– економія;

– взаємодія;

– виділення;

– символізм;

– комунікація.

11.2. Використання звуку та анімації

Щодо використання звуку як зворотного зв'язку завжди ведеться багато дискусій. Якщо все добре продумано, ненав'язливо, можна за бажанням його вмикати та вимикати, то це нормально. Перш за все слід вияснити, коли варто застосовувати звукову інформацію замість візуальної. Потім слід визначити тип звукового подання тієї чи іншої ситуації. Система повинна бути контрольованою: користувач повинен мати можливість регулювати гучність або зовсім вимикати звук.

Рекомендації щодо використання звуку, як і кольору, здебільшого основані на особливостях людського сприйняття та здатності до навчання, ніж на ПЗ [2].

Сучасні робочі середовища є відкритими, тобто користувач оточений іншими людьми, телефонами, працюючими комп'ютерами, тому *звукова форма передачі інформації малоефективна*, адже людина часто не може відрізнити, чий комп'ютер надає якісь звуки.

Більшість сучасного комп'ютерного ПЗ використовує звуковий зворотний зв'язок – короткі «біпи», які сигналізують про помилку чи неправильно обраний варіант. Але навіть такі незначні звуки можуть набридати.

Анімації, як і звуку, однакову увагу приділяють і розробники, і користувачі.

При переміщенні миші по різних областях екрана її вказівник змінює свою форму для відображення типу дії, яку ви можете виконати над цим об'єктом або в цій області.

Є багато форм, які може приймати вказівник миші (стрілка, переміщення, розмір, очікування, заборона).

Іконки і вказівники з анімацією теж можуть бути корисними.

Під анімацією розуміють зміну в часі візуального подання графічного елемента. Аналогічно звуку головна перевага анімації полягає саме в розважальності дії [2].

Курсор з анімацією спрощує його пошук на екрані, особливо це важливо для невеликих комп'ютерних записників з малим розділенням. *Анімація може використовуватись для вдосконалення візуального зв'язку між комп'ютерами та користувачами.*

Анімація може застосовуватись для виділення важливих іконок, відображення стану певного об'єкта і навіть пояснення його поведінки.

11.3. Термінологія та міжнародне проектування. Ключові питання розробки

Розробники не завжди говорять однією мовою з користувачами. У цьому і криється причина того, чому вони рідко спілкуються одне з одним – у них просто мало спільного! Ось чому розробники повинні визначити термінологію і відповідати за кожне слово, яке з'являється на екрані. Треба бути особливо уважними при організації інформації в міжнародних інтерфейсах [4–8].

До *ключових питань* розробки можна віднести:

– керуючі елементи;

– рядок меню та панель інструментів;

– метод drag-and-Drop;

– компонування і розробку вікна.

В очах користувача кожна деталь, кожен керуючий елемент повинен мати якусь мету.

З погляду розробника кожна частина даних може бути подана декількома способами.

Оптимальність керуючих елементів для збирання певних типів інформації, яку вводить користувач, залежить від типу даних, які потрібно зібрати, кількості доступних даних, вигляду подання іншої інформації на екрані, а також способу взаємодії користувачів з керуючими елементами екрана.

Одним з *ключових чинників*, який слід ураховувати при виборі керуючих елементів, *є масштабованість*.

Рядки меню, панелі інструментів і кнопки – всі вони можуть використовуватись для подання аналогічних дій.

На рис. 11.2 наведено зміст рекомендацій, які необхідно виконувати для підвищення ефективності розробки меню та панелей інструментів.

Рис. 11.2. Зміст рекомендацій, які необхідно виконувати для підвищення ефективності розробки меню та панелей інструментів

Інтерфейси стають усе більш *об'єктно-орієнтованими*, більш *графічними і візуальними*, тому перевага все частіше надається прямому маніпулюванню drag-and-drop. Проблема цього методу полягає в тому, що немає візуальної вказівки на те, що об'єкти можуть або повинні бути переміщені або скинуті на інші об'єкти [2].

Компонування і розробка елементів вікна – мистецтво і наука одночасно. Колір, шрифт, розмір, тип керуючих елементів, їх розмір, орієнтування керуючих елементів, питання симетричності, виділення, багато інших чинників – все впливає на кінцевий вигляд навіть найпростішого вікна [2].

11.4. Додаткові рекомендації до розробки КІ

На рис. 11.3 наведено перелік та зміст додаткових рекомендацій з розробки користувацького інтерфейсу.

Рис. 11.3. Перелік та зміст додаткових рекомендацій з розробки користувацького інтерфейсу

На рис. 11.4 наведено зміст основних порад, виконання яких підвищує ефективність розробки користувацького інтерфейсу.

Рис. 11.4. Зміст основних порад, виконання яких підвищує ефективність розробки користувацького інтерфейсу

Контрольні запитання до розділу 11

1. Охарактеризуйте інструментарій розробника.
2. Опишіть передачу інформації візуальним способом.
3. Опишіть використання звуку в інтерфейсі.
4. Опишіть використання анімації.
5. Наведіть термінологію та охарактризуйте міжнародне проектування.
6. Охарактризуйте ключові питання розробки.
7. Наведіть додаткові рекомендації до розробки КІ.

БІБЛІОГРАФІЧНИЙ СПИСОК

1. Уткіна Г. А. Людино-машинний інтерфейс: навч. посіб. Київ: КЕІ ДВНЗ «КНЕУ імені Вадима Гетьмана», 2011. 162 с.

2. Шевчук Р. П. Сучасні інструментальні засоби розробки користувацького інтерфейсу: опорний конспект лекцій для студентів за спеціальностями 7.05010301 «Програмне забезпечення систем», 8.05010301 «Програмне забезпечення систем». Тернопіль, 2012. 103 с.

3. Захист графічного інтерфейсу користувача (GUI) в Сінгапурі: URL: https://www.taylorvinters.com/article/protecting-your-graphical-user-interface-gui-in-singapore.

4. Baecker Ronald M., Jonathan Giirdiu, William A.S. Buxton, Saul Greenberg. Readings in Human-Computer Interaction. San Francisco, 2000. 456 p.

5. Boling Elizabeth. Usability testing for Web sites. Bloomington, 1995. 178 p.

6. Del Galdo, Elisa and Jacob Nielsen. International user interface. New York, 1996. 638 p.

7. Femandes Tony. Global Interface Design. Chestnut Hill, 1995. 871 p.

8. Hoiton William. Lee Taylor, Arthur Ignacio. Nancy Hoft. The Web Page Design Cookbook: All the Ingredients You Need to Create - star Web pages. New York, 1996. 345 p.

Навчальний посібник

Доценко Сергій Ілліч

ЛЮДИНО-МАШИННИЙ ІНТЕРФЕЙС

Відповідальний за випуск Доценко С. І.

Редактори Еткало О. О.

Підписано до друку 29.03.2021 р.

Формат паперу 60х84 1/16. Папір писальний.

Умовн.-друк. арк. 8.5. Тираж 50. Замовлення №

Видавець та виготовлювач Український державний університет
залізничного транспорту,
61050, Харків-50, майдан Фейєрбаха, 7.
Свідоцтво суб'єкта видавничої справи ДК № 6100 від 21.03.2018 р.