МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Київський національний університет будівництва і архітектури

О.В. Горда

ЕРГОНОМІКА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Конспект лекцій
для студентів спеціальностей
122 «Комп`ютерні науки»,
126 "Інформаційні системи і технології" та
015.10 "Професійна освіта. Комп`ютерні технології"

Київ 2020

УДК 517
Б39

Рецензент: О.О. Терентьєв, д-р техн. наук, професор

Затверджено на засіданні вченої ради факультету автоматизації і інформаційних технологій, протокол №6 від 19 лютого2020 року.

Горда О.В.
Б39 Ергономіка інформаційних технологій: конспект лекцій / О.В. Горда. - К.: КНУБА, 2020. - 83с.

Метою дисципліни є внесення ергономічних знань в освіту студентів комп'ютерних спеціальностей. В конспекті лекцій розглянуто поняття ергономіки та її значення у сучасному суспільстві, зокрема для інформаційних технологій які сьогодні знайшли впровадження у всі галузі діяльності. Розглянуті основні напрямки ергономічних досліджень та визначенні їх характеристики та особливості застосування саме для інформаційних технологій. Особлива увага приділена розробці ергономічних програмних додатків з точки зору користувача. Конспект лекцій містить необхідний теоретичний і довідковий матеріал, контрольні запитання, список літератури.

Призначений для студентів, які навчаються за спеціальністю 122 «Комп`ютерні науки», 126 "Інформаційні системи і технології", 015.10 "Професійна освіта. Комп`ютерні технології"

УДК 517
Б39

Зміст

Вступ

Дисципліна «Ергономіка інформаційних технологій» входить до складу циклу підготовки бакалаврів спеціальностей 122 «Комп`ютерні науки», 126 "Інформаційні системи і технології", 015.10 "Професійна освіта. Комп`ютерні технології".

Метою дисципліни є внесення ергономічних знань в освіту студентів комп'ютерних спеціальностей.

Впровадження інформаційних технологій (ІТ), засобів автоматики підвищує роль людини у застосуванні людино-машинних систем. Оптимізація інформаційного завантаження людини-оператора є, на сьогоднішній день, однією з основних проблем ергономіки. У зв'язку з цим, її найважливішим завданням є проектування інформаційної моделі системи взаємодії користувача з програмним продуктом.

Фахівці-системотехнік повинні володіти навичками проектування призначених для користувача інтерфейсів, вміти проводити дослідження зручності використання і ергономічну експертизу інформаційних систем різної складності і призначення. Розробник програмного додатку повинен враховувати той факт, що інтерфейс користувача інформаційної системи повинен задовольняти ряду критеріїв: мінімальний час виконання завдання користувачем; мінімальне число мимовільних помилок користувача; мінімальна неоднозначність в розумінні інтерфейсу (що сприяє самонавчанню користувачів і робить їх поведінку передбачуваною); висока стандартизація інтерфейсу (полегшує навчання користувачів); обсяг введеної користувачем інформації повинен прагнути до мінімуму; простота і візуальна привабливість.

Конспект лекцій відповідає програмі дисципліни і допоможе освоїти основні принципи проектування користувальницького інтерфейсу інформаційної системи.

Особливістю конспекту лекцій є вивчення замкнутого циклу проектування призначеного для користувача інтерфейсу інформаційної системи.

Викладений матеріал проілюстрований схемами та прикладами проектування інтерфейсу користувача. До кожної лекції наведений перелік контрольних питань.

Лекція 1. Ергономіка як наука і її місце в інформаційних технологіях

1.1. Визначення поняття «ергономіка»

Ергономіка є системно-орієнтованою дисципліною, яка в даний момент охоплює всі аспекти людської діяльності. Ергономіка розвиває цілісний підхід, що поєднує розгляд і облік фізичних, когнітивних (розумових), соціальних, організаційних та інших значущих чинників. Практикуючий ергономіст повинен мати широку ерудицію у всіх цих сферах. Ергономісти часто працюють в конкретних секторах або предметних областях, які постійно еволюціонують – створюються нові, а старі отримують нові перспективи розвитку.

Ергономіка (від грец. Ἔργον - робота + νόμος – закон) – в традиційному розумінні – наука про пристосування посадових обов'язків, робочих місць, предметів і об'єктів праці, а також комп'ютерних програм для найбільш безпечного та ефективного праці працівника, виходячи з фізичних і психічних особливостей людського організму.

Більш широке визначення ергономіки, прийняте в 2010 році Міжнародною асоціацією ергономіки (англ.), Звучить так: «Наукова дисципліна, що вивчає взаємодію людини та інших елементів системи, а також сфера діяльності по застосуванню теорії, принципів, даних і методів цієї науки для забезпечення благополуччя людини та оптимізації загальної продуктивності системи ».

Усередині ергономіки є напрямки, які більш глибоко вивчають специфічні особливості людини і характеристики його взаємодії. Сьогодні в ергономіці можна виділити наступні області:

- *Фізична ергономіка* розглядає анатомічні, антропометричні, фізіологічні і біомеханічні характеристики і їх вплив на фізичну діяльність людини. До питань цього напряму належать робочі пози, вантажно-розвантажувальні роботи, монотонні рухи, робота, яка загрожує м'язово-скелетними розладами, компоновка робочого місця, безпеку і здоров'я.

- *Когнітивна ергономіка* пов'язана з розумовими процесами, такими як сприйняття, пам'ять, міркування, моторна реакція і їх роллю у взаємодії людини з іншими елементами системи. Цей напрямок вивчає розумове навантаження, процеси прийняття рішення, роботу, що вимагає високої кваліфікації, взаємодія людини з комп'ютером, надійність людини, професійний стрес і професійну підготовку.
- *Організаційна ергономіка* націлена на оптимізацію соціотехнічних систем, включаючи їх організаційну структуру, політику і процеси. Питаннями організаційної ергономіки є комунікація, управління трудовими ресурсами, проектування діяльності, проектування робочого часу, колективна робота, нові парадигми організації праці, віртуальні організації, віддалена робота і управління якістю.

Предметом ергономіки як науки є вивчення системних закономірностей взаємодія людини (групи людей) з технічними засобами, об'єктом діяльності та середовищем в процесі досягнення мети діяльності або при спеціальній підготовці до її виконання у трудовій сфері та відпочинку.

Мета ергономіки – підвищення ефективності та якості діяльності людини в системі | «Людина-машина-об'єкт діяльності-середовище» (скорочено «людина- машина-середовище») при одночасному збереженні здоров'я людини і створенні передумов для розвитку його особистості.

Об'єктом дослідження в ергономіці є система «людина - машина - середовище», іншими словами досліджується взаємозв'язки людини з предметним світом в процесі трудової та інших видів діяльності. Але можуть розглядатися й інші системи, наприклад, система взаємодії людей у виробничому або іншому колективі.

Завданням ергономіки як сфери практичної діяльності є проектування та удосконалення процесів (способів, алгоритмів, прийомів) виконання діяльності і способів спеціальної підготовки (навчання, тренування, адаптації) до неї, а також тих характеристик засобів та умов, які безпосередньо впливають на ефективність і якість діяльності і психофізіологічний стан людини.

Ергономічні вимоги – це вимоги, які пред'являються до системи «людина-машина - середовище» з метою оптимізації діяльності людини-оператора з урахуванням егосоціально-психологічних, психофізіологічних, психологічних, антропологічних, фізіологічних та інших об'єктивних характеристик і можливостей. Ергономічні вимоги є основою при формуванні конструкції машини, дизайнерської розробки, просторово-композиційних рішень системи в цілому і окремих її елементів.

Людина-оператор – будь-яка людина, що керує машиною: диспетчер аеропорту, робітник-верстатник, домогосподарка біля плити або з пилососом і т.д. – для ергономіста всі вони являються операторами. Ергономіка і її методи останнім часом все ширше використовуються при проектуванні не тільки технічних пристроїв, але і архітектурних об'єктів, інтер'єрів, елементів їх обладнання. Тому доцільно замість поняття «машина» вживати більш узагальнені поняття «виріб», «предмет», а замість Система – поєднання взаємодіючих чинників, компонентів, об'єднаних певною єдиною метою, системність –властивість системи.

Машина або інструмент діяльності (виріб, предмет) в ергономіці – будь-який технічний пристрій, призначений для цілеспрямованої зміни матерії, енергії, інформації та ін. Поняття «машина» може означати як самі прості засоби (ніж, молоток і т.п.), так і складні – верстати, ЕОМ або космічні кораблі.

Ергономічні властивості – це властивості виробів (машин, предметів або їх сукупностей), які проявляються в системі «людина-машина (предмет) - середовище» в результаті реалізації ергономічних вимог.

Блок оперативних засобів і методів охоплює три найважливіші напрямки ергономічних досліджень об'єкта «людина-предмет-середовище»:

- аналіз,
- синтез (моделювання) і
- оцінка об’єкта.

Результати ергономічного дослідження – науково і експериментально обґрунтовані дані, необхідні для проектної розробки системи.

1.2. Процес ергономічного супроводу

Процес проектування системи з самого початку повинен бути орієнтований на формування її (системи) ергономічних властивостей як на одну з найважливіших цілей, що досягаються в процесі ергономічного забезпечення проектування. Весь процес ергономічного супроводу (забезпечення) проектування можна представити у вигляді наступних етапів:

- аналіз діяльності людини з дослідженням факторів її протікання;
- розробка ергономічних вимог і показників, а також рекомендацій щодо їх обліку;
- формування ергономічних властивостей проектованої техніки (вироби) і середовища;
- заключний етап – оцінка повноти і правильності реалізації ергономічних вимог (ергономічна оцінка і атестація).

Ергономіка органічно пов'язана з дизайном, однією з головних цілей якого є формування гармонійної предметної середовища, що відповідає матеріальним і духовним потребам людини. При цьому відпрацьовуються не тільки властивості зовнішнього вигляду предметів, але, головним чином, їх структурні зв'язки, які надають системі функціональне і композиційне єдність (з точки зору як виробника, так і споживача). Саме остання обставина дозволяє розглядати ергономіку як природничу основу дизайну. У практичному плані облік людських чинників – невід'ємна частина процесу дизайнерського проектування.

В ергономіці ведуться пошуки взаємного пристосування техніки і людини: з одного боку адаптація техніки до людських можливостей, з іншого – пристосування людини до умов праці. Ергономічний підхід до вирішення задачі оптимізації життєдіяльності людини визначається комплексом чинників. Головні з них, обумовлені індивідуальними особливостями людини, наведені нижче.

1. Соціально-психологічні чинники припускають відповідність конструкції машини (обладнання, оснащення) та організації робочих місць характеру і ступеня групової взаємодії, а також встановлюють характер міжособистісних відносин, що залежить від змісту спільної діяльності з управління об'єктом.

2. Антропометричні чинники обумовлюють відповідність структури, розмірів обладнання, оснащення та їх елементів структурі, формі, розмірам і масі людського тіла, відповідність характеру форм вироби анатомічної пластиці людського тіла.

3. Психологічні чинники зумовлюють відповідність обладнання, технологічних процесів і середовища можливостям і особливостям сприйняття, пам'яті, мислення, психомоторики закріплених і знову формуються навичок працюючої людини.

4. Психофізіологічні фактори обумовлюють відповідність обладнання зорових, слухових та інших можливостей людини, умов візуального комфорту і орієнтування в предметному середовищі.

5. Фізіологічні фактори покликані забезпечити відповідність обладнання фізіологічним властивостям людини, його силовим, швидкісним, біомеханічних і енергетичними можливостями.

6. Гігієнічні (гігієна – грец. Hyhieinos – приносить здоров'я) фактори зумовлюють вимоги до освітленості, газовому складу повітряного середовища, вологості, температури, тиску, запиленості, вентиляції, токсичності, напруженості електромагнітних полів, різних видів випромінювань, в т.ч. радіації, шуму (звуку), ультразвуку, вібрацій, гравітаційної перевантаження та прискоренню.

Базовими для ергономіки є психологічні моменти, пов'язані, перш за все, з психологією праці. Основні серед них такі:

- психологічні особливості особистості;
- психологічні особливості уваги;
- роль психологічного клімату в колективі.

1.3. Ергономіка в інформаційних технологіях.

Ергономіку в інформаційних технологіях можна розглядати в трьох основних аспектах:

1) ергономіка робочого місця оператора або програміста;
2) ергономіка розробки програмного забезпечення (ергономіка для програміста);
3) ергономіка використання програмних розробок (ергономіка користувача).

1.3.1 Ергономіка робочого місця

Основний принцип організації робочого місця – мінімізація навантажень, зручність і комфортність. Характеристика ергономіки робочого місця визначається психологічними, фізіологічними і антропометричними вимогами. Відповідно до цього враховується робоча поза; можливість охоплення рухами і поглядом всього простору та розташованих на ній предметів; простір, на якому розміщується сам працівник; можливість роботи з технікою, ведення записів, розміщення необхідних матеріалів.

Відповідно до вимог ергономіки, комп'ютер повинен розташовуватися в приміщенні так, щоб світло з вікна падало зліва, штучне освітлення має бути рівномірним, конструкція робочого столу 727 мм, стілець підйомно-поворотний, верхня частина екрану повинна розташовуватися на рівні очей, відстань від монітора до очей 60-80 см, поверхня клавіш увігнута, довжина простору для ніг 450 мм, висота простору для ніг 600мм, ширина простору для ніг 500 мм.

Характеристика робочого місця з ПК будь-якого фахівця повинна відповідати певним правилам і враховувати: антропометричну сумісність – відповідність розмірів тіла і його положення при роботі; сенсомоторну сумісність – швидкість моторних операцій; енергетичну сумісність – зусилля, прикладені для виконання дій; психофізіологічну сумісність – реакцію на зовнішні естетичні параметри. Крім форми і габаритів меблів, розташування людини щодо техніки і розміщення клавіатури, монітора і інших частин робочого процесу, ергономіка робочого місця за комп'ютером повинна враховувати розміри і форму периферії. Основні ергономічні вимоги організації робочого місця з домашнім ПК наведені у табл. 1.1.

Таблиця 1.1.

Оцінка відповідності організації робочого місця з домашнім ПК вимогам ергономіки

Характеристика	***Вимоги ергономіки***	***Фізичне значення***
Штучне світло	Розподіляється рівномірно	Розподіляється рівномірно
Світло з вікна	Розташовується ліворуч	Розташовується праворуч

Продовження таблиці 1.1.

Характеристика	***Вимоги ергономіки***	***Фізичне значення***
Ширина простору для ніг	500 мм	561 мм
Довжина простору для ніг	450 мм	512 мм
Стілець	Підйомно-поворотний	Звичайний
Відстань від очей до монітора	60-80 см	62 см
Верхня частина екрану	Розташовується на рівні очей	Розташовується вище очей
Поверхня клавіш	Увігнута	Увігнута
Висота простору для ніг	600 мм	596 мм
Конструкція робочого столу	727 мм	723 мм

Слід зазначити, що при невиконанні ергономічних вимог розвиваються безліч хвороб симптомами яких є біль і дискомфорт в попереку, які виникають через неправильне положення спини, сутулості, неправильного положення ніг; через неправильне положення рук на клавіатурі або миші можуть серйозно постраждати кисті, зап'ястя і передпліччя (найпоширенішим захворюванням є кистьовий тунельний синдром); доводиться нахиляти спину, шию при роботі з монітором і документами, що призводить до підвищених навантажень і напруг м'язів, і викликає біль і дискомфорт м'язів спини, шиї і плечової частини корпусу.

Таким чином, якщо буде врахована ергономіка при роботі за комп'ютером, правильно б лаштоване робоче місце і простір робочого місця, зникнуть деталі, що доставляли дискомфорт і заважали роботі будь-якого фахівця.

Такі зміни дозволять як збільшити ефективність, так і заощадять час, який витрачався на самостійне подолання нехай навіть непомітних відразу, але важливих «дрібниць».

1.3.2 Ергономіка розробки програмного забезпечення

При розробці програмного забезпечення необхідно брати до уваги, що воно створюється як правило колективом. Для забезпечення ефективної взаємодії членів колективу та розробки програмного продукту необхідно:

- дотримуватись встановлених стандартів;
- дотримуватись вимог до програмного забезпечення;
- доцільно застосовувати CASE-технології.

Міжнародні стандарти проектування інформаційних систем ISO / IEC 12207 – базовий стандарт на процеси життєвого циклу ІС, він орієнтований на різні типи проектів. У стандарті не передбачено конкретних етапів життєвого циклу ІС. Замість того визначені тільки ряд процесів. Тому стандарт дозволяє реалізувати довільну модель життєвого циклу, і це є його головною перевагою.

Тому для побудови повноцінної системи якості, заснованої на моделі ISO, необхідно використовувати велику кількість допоміжних галузевих і ISO стандартів.

Стандарт TickIT

Досить широку популярність здобув британський стандарт TickIT. Цей галузевий стандарт регламентує вимоги до системи якості для організацій розробників програмного забезпечення і базується на моделі ISO 9001: 94. На відміну від моделі ISO 9001, яка регламентує "що необхідно зробити", розробники даного стандарту спробували відповісти на питання "як" можна виконати вимоги, визначені в ISO 9001. TickIT об'єднує в собі модель ISO 9001 з набором рекомендаційних стандартів ISO 12207 та ISO 9000 -3.

Стандарти SEI SW-CMM

Дуже цікавий підхід до поліпшення внутрішніх процесів розробки програмного забезпечення визначено у моделі SEI SW-CMM. В основу даної моделі (також як і в основу стандартів ISO серії 9000) покладена теорія TQM. Теорія TQM ґрунтується на поступовому поліпшенні внутрішніх виробничих процесів за рахунок безлічі невеликих впроваджуваних в компанії поліпшень. Однак, моделі ISO і CMM не однакові в своїх підходах до побудови

самовдосконалюються систем управління якістю і поліпшення виробничих процесів.

На відміну від моделі ISO, де для того, щоб відповідати вимогам, необхідно продемонструвати 100% -ву відповідність моделі (і тільки воно дозволяє компанії самовдосконалюватися), в моделі SEI SW-CMM передбачено поетапний підхід до побудови системи вдосконалення процесів. Для досягнення цієї мети розробники стандарту CMM визначили п'ять рівнів, які повинна пройти організація для того, щоб досягти основної мети - підвищення ефективності функціонування процесів компанії і, як наслідок, поліпшення якості результатів виробничих процесів і розроблюваного програмного забезпечення.

Стандарти по Project Management

Одним з важливих моментів, який необхідно мати на увазі при впровадженні будь-яких стандартів (ISO 9000, SEI SW-CMM, TickIT, Spice ISO 15504 тощо), пов'язаний з тим, що структура виробництва компаній, що розробляють програмне забезпечення, пов'язана зі специфікою продукту. Кожен продукт, що розробляється IT-компанією, унікальний. І для його розробки, як правило, використовується проектний тип організації виробництва, який тісно пов'язаний з матричною структурою управління проектами.

Вимоги до програмного забезпечення – сукупність тверджень щодо атрибутів, властивостей або якостей програмної системи, яка підлягає реалізації. Створюються в процесі розробки вимог до програмного забезпечення, в результаті аналізу вимог.

Вимоги можуть виражатися у вигляді текстових тверджень і графічних моделей.

У класичному технічному підході сукупність вимог використовується на стадії проектування ПО. Вимоги також використовуються в процесі перевірки ПО, так як тести ґрунтуються на певних вимогах.

Етапу розробки вимог, можливо, передує техніко-економічне обґрунтування, або концептуальна фаза аналізу проекту. Фаза розробки вимог може бути розбита на виявлення вимог (збір, розуміння, розгляд та з'ясування потреб зацікавлених осіб), аналіз

(перевірка цілісності і закінченості), специфікація (документування вимог) і перевірка правильності.

Види вимог за рівнями.

- Бізнес-вимоги – визначають призначення ПО, описуються в документі про бачення (vision) і межах проекту (scope).
- Вимоги користувача – визначають набір призначених для користувача завдань, які повинна вирішувати програма, а також способи (сценарії) їх рішення в системі. Призначені для користувача вимоги можуть виражатися у вигляді фраз тверджень, у вигляді способів застосування (use case), призначених для користувача історій (user story), сценаріїв взаємодії (scenario).
- Функціональні вимоги – визначають «як» реалізувати продукт. Описується в системній специфікації (system requirement specification, SRS).

Види вимог за характером.

- Функціонального характеру – вимоги до поведінки системи.
  - Бізнес-вимоги.
  - Вимоги для користувача.
  - Функціональні вимоги.
- Не функціонального характеру – вимоги до характеру поведінки системи.

- Бізнес-правила – визначають обмеження, що виникають з предметної області і властивостей об'єкта, що автоматизується (підприємства).

- Системні вимоги і обмеження – визначення елементарних операцій, які повинна мати система, а також різних умов, яким вона може задовольняти.

  До системних обмежень відносяться обмеження на програмні інтерфейси, вимоги до атрибутів якості, вимоги до вживаного обладнання та програмного забезпечення.
  - Атрибути якості.
  - Зовнішні системи і інтерфейси.
  - Обмеження.

Джерела вимог

- Державне, місцеве і галузеве законодавство (конституція, закони, розпорядження).
- Нормативне забезпечення організації (регламенти, положення, статути, накази).
- Поточна організація діяльності об'єкта автоматизації.
- Моделі діяльності (діаграми бізнес-процесів).
- Уявлення і очікування споживачів і користувачів системи.
- Журнали використання існуючих програмно-апаратних систем.
- Конкуруючі програмні продукти.

Характеристики якісних вимог

Характеристики якісних вимог по-різному визначені різними джерелами, однак, вони є загальновизнаними (табл.. 1.2).

Таблиця 1.2.

Характеристики якісних вимог

Характеристика	**Пояснення**
Одиничність	Вимога описує одну і тільки одну річ.
Завершеність	Вимога повністю визначена в одному місці і вся необхідна інформація присутня.
Послідовність	Вимога не суперечить іншим вимогам і повністю відповідає зовнішній документації.
Атомарність	Вимога «атомарна». Тобто вона не може бути розбита на ряд більш детальних вимог без втрати завершеності.
Відстеження	Вимога повністю або частково відповідає діловим потребам як заявлено зацікавленими особами і документовано.
Актуальність	Вимога не стала застарілою з плином часу.
Здійсненність	Вимога може бути реалізована в межах проекту.

Продовження таблиці 1.2.

Характеристика	Пояснення
Недвозначність	Вимога коротко визначена без звернення до технічного жаргону і прихованим формулювання, виражає об'єктивні факти, а не суб'єктивні думки. Можлива одна і тільки одна інтерпретація. Визначення не містить нечітких фраз. Використання негативних тверджень і складових тверджень заборонено.
Обов’язковість	Вимога представляє певну характеристику, відсутність якої призведе до неповноцінності рішення, яка не може бути проігнорована.
Верифікованість	Реалізованість вимоги може бути визначена через один з чотирьох можливих методів: огляд, демонстрація, тест чи аналіз.

Засто***сування CASE-засобів.*** На перших етапах розробки програмного продукту виконується його проектування і тільки після завершення проектування переходять до написання програм – кодування. Для полегшення процесу проектування та переходу до кодування сьогодні широко застосовуються CASE-засоби.

CASE (англ. Computer-aided software engineering) – набір інструментів і методів програмної інженерії для проектування програмного забезпечення, який допомагає забезпечити високу якість програм, відсутність помилок і простоту в обслуговуванні програмних продуктів. Також під CASE розуміють сукупність методів і засобів проектування інформаційних систем з використанням CASE-інструментів.

Засоби автоматизації розробки програм (CASE-засобу) – інструменти автоматизації процесів проектування і розробки програмного забезпечення для системного аналітика, розробника ПЗ і програміста. Спочатку під CASE-засобами розумілися тільки інструменти для спрощення найбільш трудомістких процесів аналізу і проектування, але з приходом стандарту ISO / IEC 14102 CASE-засоби

стали визначати, як програмні засоби для підтримки процесів життєвого циклу програмного забезпечення (ПЗ).

Основною метою CASE-технології є розмежування процесу проектування програмних продуктів від процесу кодування і наступних етапів розробки, максимально автоматизувати процес розробки. Для виконання поставленої мети CASE-технології використовують два принципово різних підходи до проектування: структурний і об'єктно-орієнтований.

Структурний підхід передбачає декомпозицію (поділ) поставленого завдання на функції, які необхідно автоматизувати. У свою чергу, функції також розбиваються на підфункції, завдання, процедури. В результаті виходить впорядкована ієрархія функцій і переданої інформацією між функціями.

Структурний підхід має на увазі використання певних загальноприйнятих методологій при моделюванні різних інформаційних систем:

SADT (structured analysis and design technique);

DFD (data flow diagrams);

ERD (entity-relationship diagrams).

Існує три основних типи моделей, що використовуються при структурному підході: функціональні, інформаційні та структурні.

Основним інструментом об'єктно-орієнтованого підходу є мова UML – уніфікована мова моделювання, який призначений для візуалізації та документування об'єктно-орієнтованих систем з орієнтацією їх на розробку програмного забезпечення. Дана мова включає в себе систему різних діаграм, на підставі яких може бути побудовано уявлення про проектованої системі.

CASE-технології мають ряд характерних особливостей:

- мають графічні засоби для проектування і документування моделі інформаційної системи
- мають організоване спеціальним чином сховище даних, що містить інформацію про версії проекту і його окремих компонентах
- розширюють можливості для розробки систем за рахунок інтеграції декількох компонент CASE-технологій

Сучасні CASE-засоби підтримують також безліч технологій моделювання інформаційних систем, починаючи від простих методів аналізу і регламентації і закінчуючи інструментами повної автоматизації процесів всього життєвого циклу програмного забезпечення. Найбільш широко вони застосовуюь при розробці програмних продуктів на основі компонентного підходу, що лежить в основі технологій, розроблених на базі COM (Component Object Model – компонентна модель об'єктів), і технології створення розподілених додатків CORBA (Common Object Request Broker Architecture – загальна архітектура з посередником обробки запитів об'єктів). Ці технології використовують подібні принципи і розрізняються лише особливостями їх реалізації.

CASE-технології можна класифікувати за функціональної спрямованості на

- засоби моделювання предметної області;
- засоби аналізу і проектування;
- технології проектування схем баз даних;
- засоби розробки додатків;
- технології реінжинірингу програмного коду і схем баз даних.

На даний момент на ринку програмного забезпечення налічується понад 300 різних CASE-засобів. Найбільш відомими є CA ERwin Process Modeler (раніше BPwin), CA ERwin Data Modeler (раніше ERwin), Rational Rose, ARIS і т. д.

Процес успішного застосування CASE-засобів не обмежується лише їх використанням. Він охоплює реалізацію та планування багатьох організаційних, планувальних, технічних та структурних процесів та зміну у загальній культурі застосування автоматизованих інформаційних технологій.

Основні аспекти ергономіки робочого місця та розробки програмного забезпечення розглядаються в інших дисциплінах, як , наприклад, «охорона праці» та «розробка систем». Наша дисципліна буде розглядати основні ергономічні аспекти взаємодії користувача з програмним продуктом, а саме розробці ефективних інтерфейсів або фронтенд програмуванню.

Контрольні питання.

1. Що таке ергономіка? Яке сучасне визначення цього поняття?
2. Що є предметом та об’єктом науки ергономіки?
3. Яка основна мета ергономіки?
4. Яке основне завдання ергономіки?
5. Якими основними поняттями оперує наука ергономіка?
6. З яких процес етапів складається процес ергономічного супроводу (забезпечення) проектування?
7. Якими чинниками визначається ергономічний підхід до вирішення задачі оптимізації життєдіяльності людини?
8. Які психологічні моменти, пов'язані, перш за все, з психологією праці?
9. Які основні фактори людини враховуються при оцінці ергономічності системи?
10. В яких аспектах можна розглядати ергономіку в інформаційних технологіях?
11. Що включає ергономіка автоматизованого робочого місця(АРМ).
12. Яким основним характеристикам повинно задовольняти робоче місце програміста?
13. На які стандарти спирається процес розробки програмних продуктів?
14. Які основні стандарти розробки програмних продуктів Ви знаєте?
15. Що таке вимоги до програмного продукту?
16. Які типи вимог до програмних продуктів Ви знаєте?
17. Яким характеристикам повинні задовольняти вимоги до програмних продуктів та їх розробки?
18. Що таке CASE-технології?
19. Які сучасні CASE-засоби Ви знаєте?
20. Де найчастіше застосовуються CASE-технології?
21. Які переваги дає використання CASE-технології розробникам?

Лекція 2. Ергономіка користувача та структура програмного забезпечення

2.1. Вимоги до програмного забезпечення основні підходи до його проектування

На сьогоднішній день технічне оснащення сучасних робочих місць вимагає ретельного освоєння комп'ютерних програм як основних засобів праці. Взаємодія з комп'ютером стає невід'ємною частиною їх роботи.

У зв'язку з цим тема ергономіки в проектуванні програмного забезпечення і в, зокрема, інтерфейсів є актуальною і затребуваною.

З точки зору ергономіки, цілі, які необхідно досягти в результаті проектування і розробки програмних продуктів наступні:

- Ефективність: за допомогою даного програмного продукту користувач може досягти цілей і вирішити завдання, які перед ним стоять, з мінімальними затратами.
- Продуктивність, результативність: за мінімальний час досягати найвищого результату.
- Суб'єктивна задоволеність працею користувача: відсутність роздратування, невдоволення, негативних емоцій.

Досягнення зазначених цілей значною мірою залежить від якості запроектованої архітектури програмного забезпечення.

Під час розробки архітектури програмного забезпечення виконується його модульно-ієрархічна побудова.

Модуль – це замкнута програма, яку можна викликати з будь-якого іншого модуля в програмі і можна окремо компілювати.

Прагнення до створення модульних програм пояснюється наступними факторами:

- програмні модулі, як правило, вирішують невелику функціональне завдання, використовують на вході і на виході трохи даних. Внутрішні змінні модуля не пов'язані з внутрішніми змінними інших модулів. Тому окремі модулі можуть створюватися і регламентуватиме різними розробниками незалежно один від одного;

- модульні програми легко читати, супроводжувати і модифікувати. Виправлення окремого модуля викликає мінімальні зміни в інших модулях, пов'язаних з ним по управлінню та інформації;
- модульні програми мають підвищену надійністю, так як при їх розробці існує можливість розподілу робіт по створенню модулів різної складності і важливості між програмістами різної кваліфікації;
- можна створювати бібліотеки найбільш уживаних підпрограм, які потім можна використовувати в якості комплектуючих частин при розробці інших додатків;
- процедура завантаження всієї програми в оперативну пам'ять спрощується при використанні оверлейного методу;
- виникає багато природних контрольних точок для спостереження за просуванням проекту з управління і за інформацією.

Для розробки архітектури застосовується атрибутний метод проектування (Attribute-Driven Design, ADD) – це методика визначення програмної архітектури, в якій процес декомпозиції ґрунтується на передбачуваних атрибутах якості продукту. Це рекурсивний процес декомпозиції, на кожному з етапів якого відбувається відбір тактик і архітектурних зразків, які відповідають тим чи іншим сценаріями якості, а також розподіл функціональності, спрямоване на конкретизацію типів модулів даного зразка.

Результатом застосування ADD є перші кілька рівнів подання декомпозиції модулів архітектури, а також всі інші пов'язані з ними уявлення. Не варто, втім, думати, що після ADD стають відомі всі деталі уявлень, - система описується як набір контейнерів функціональності і існуючих між ними взаємозв'язків.

Етапи ADD:

1. Вибір модуля для декомпозиції. Як правило, в якості вихідного модуля береться система в цілому. Всі необхідні вхідні дані (обмеження, функціональні вимоги і вимоги до якості) для нього повинні бути відомі.

2. Уточнення модуля в кілька етапів:

а) вибір архітектурних мотивів з набору конкретних сценаріїв реалізації якості і функціональних вимог. На цьому етапі визначаються найбільш важливі в контексті проведення декомпозиції моменти;

б) вибір архітектурного зразка, відповідного архітектурним мотивів. Створення (або вибір) зразка, тактики якого дозволяють реалізувати ці мотиви. Виявлення дочірніх модулів, необхідних для реалізації цих тактик;

в) конкретизація модулів, розподіл функціональності з елементів Use Case, складання кількох вистав;

г) визначення інтерфейсів дочірніх модулів. Декомпозиція має своїм результатом нові модулі, а також обмеження на типи взаємодії між ними. Для кожного з модулів ці відомості слід зафіксувати в документації по інтерфейсу;

д) перевірка та уточнення елементів Use Case в сценаріях якості і накладення, виходячи з них, обмежень на дочірні вузли. На цьому етапі ми перевіряємо, чи всі фактори врахували, а також, в розрахунку на подальшу декомпозицію або реалізацію, готуємо дочірні модулі.

3. Перераховані вище етапи слід виконати щодо всього, що потребує подальшої декомпозиції модулів.

Для оцінки розробленої архітектури застосовують метод аналізу компромісних архітектурних рішень (Architecture Tradeoff Analysis Method, ATAM) – комплексна універсальна методика оцінки програмної архітектури. Відповідно до назви цей метод виявляє ступінь реалізації в архітектурі тих чи інших завдань за якістю, а також (виходячи з припущення про те, що будь-яке архітектурне рішення впливає відразу на кілька завдань за якістю) механізм їх взаємодії - іншими словами, їх взаємозамінність.

Метод аналізу вартості та ефективності (Cost Benefit Analysis Method, CBAM) базується на методі ATAM і забезпечує моделювання витрат і вигод, пов'язаних з прийняттям архітектурно-проектних рішень, і сприяє їх оптимізації. Методом CBAM оцінюються технологічні та економічні фактори, а також самі архітектурні рішення.

Всі програмні архітектори і відповідальні особи прагнуть довести до максимуму різницю між вигодами, отриманими від системи, і вартістю реалізації її проектного рішення. Контекст CBAM зображений на рис. 2.1.

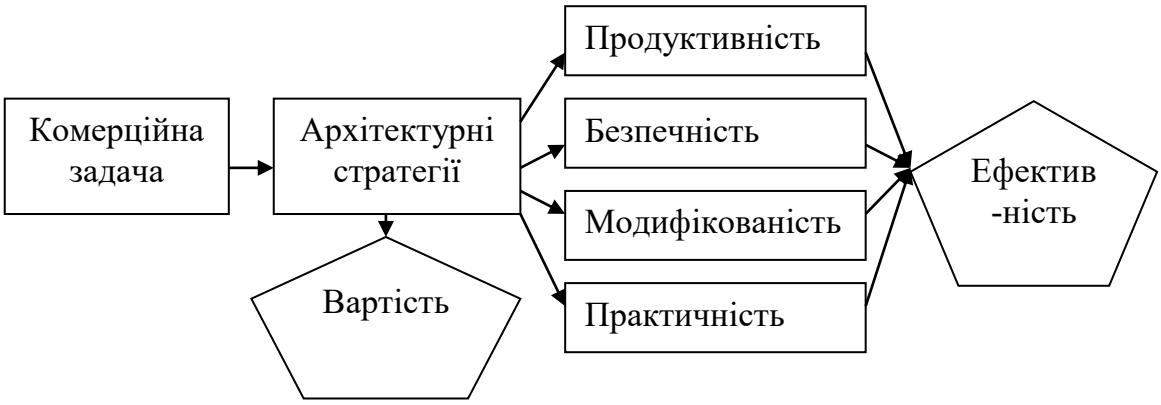


Рис. 2.1. Контекст методу аналізу вартості та ефективності (CBAM)

2.2. Архітектура програмного комплексу

Архітектура програмного забезпечення (англ. Software architecture, ПЗ) – це структура програми або обчислювальної системи, яка включає програмні компоненти, видимі зовні властивості цих компонентів, а також відносини між ними. Цей термін стосується також документування архітектури програмного забезпечення. Документування архітектури ПЗ спрощує процес комунікації між зацікавленими особами (англ. Stakeholders), дозволяє зафіксувати прийняті на ранніх етапах проектування рішення про високо рівневі дизайні системи і дозволяє використовувати компоненти цього дизайну і шаблони повторно в інших проектах.

Архітектура ПЗ є реалізацією не функціональних вимог до системи, в той час як проектування ПЗ є реалізацією функціональних вимог.

Архітектурний вигляд складається з 2 компонентів:

• елементи;

• відносини між елементами.

Структурні компоненти архітектури програмного комплексу та зв’язки між ними значною мірою визначаються характером задач для яких він призначений та вимогами майбутнього користувача або замовника.

Загальна архітектура програмного комплексу для вирішення прикладних задач представлена на рис. 2.2.

Рис. 2.2. Загальна структура програмного комплексу для вирішення прикладних задач

Архітектурні види можна поділити на 3 основні типи:

1. Модульні види (англ. Module views) – показують систему як структуру з різних програмних блоків.

2. Компоненти-і-конектори (англ. Component-and-connector views) - показують систему як структуру з паралельно запущених елементів (компонентів) і способів їх взаємодії (конекторів).

3. Розміщення (англ. Allocation views) – показує розміщення елементів системи в зовнішніх середовищах.

Приклади модульних видів (рис. 2.3., рис. 2.4.):

• Декомпозиція (англ. Decomposition view) – складається з модулів в контексті відношення «є підмодулем».

• Використання (англ. Uses view) – складається з модулів в контексті відношення «використовує» (тобто один модуль використовує сервіси іншого модуля)

• Вид рівнів (англ. Layered view) – показує структуру, в якій пов'язані за функціональністю модулі об'єднані в групи (рівні)

• Вид класів / узагальнень (англ. Class / generalization view) – складається з класів, які пов'язані через відношення «успадковується від» і «є екземпляром».

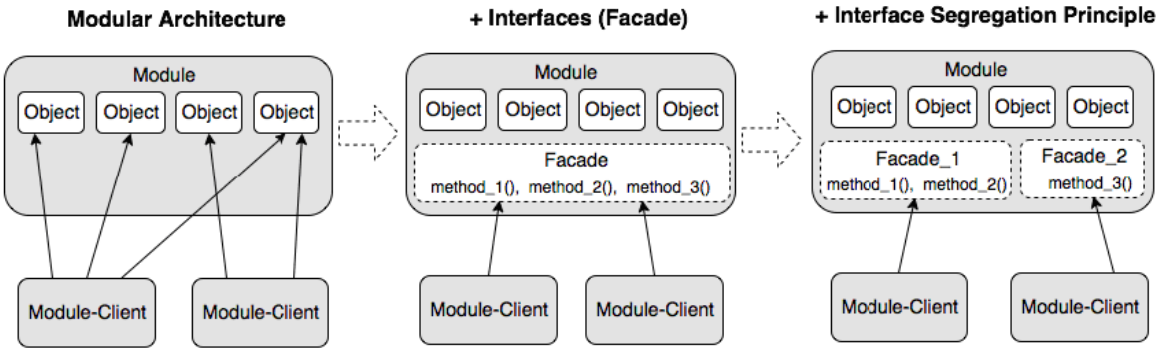


Рис. 2.3. Приклади проектування модулів

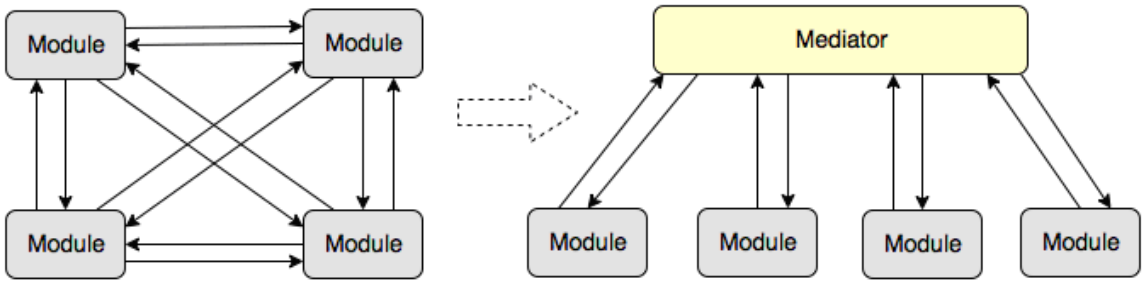


Рис. 2.4. Приклади проектування модульних структур

Приклади видів компонентів-і-конекторів:

• Процесний вид (англ. Process view) – складається з процесів, з'єднаних операціями комунікації, синхронізації і / або виключення.

• Паралельний вид (англ. Concurrency view) – складається з компонентів і конекторів, де конектори представляють собою «логічні потоки»

• Вид обміну даними (англ. Shared-data (repository) view) – складається з компонентів і конекторів, які створюють, зберігають і отримують постійні дані.

• Вид клієнт-сервер (англ. Client-server view) – складається з взаємодіючих клієнтів і серверів, а також конекторів між ними (наприклад, протоколів і спільних повідомлень)

Приклади видів розміщення:

• Розгортання (англ. Deployment view) – складається з програмних елементів, їх розміщення на фізичних носіях і комунікаційних елементів

• Впровадження (англ. Implementation view) – складається з програмних елементів і їх відповідності файловим структурам в різних середовищах (розробницького, інтеграційної та т.д.)

• Розподіл роботи (англ. Work assignment view) – складається із модулів і опису того, хто відповідальний за впровадження кожного з них.

Хоча було розроблено кілька мов для опису архітектури програмного забезпечення, в даний момент немає згоди з приводу того, який набір видів повинен бути прийнятий в якості еталону. Як стандарт «для моделювання програмних систем (і не тільки)» була створена мова UML де модульна структура представляється діаграмою пакетів.

2.3. Структура програми з точки зору користувача

Людино-машинний інтерфейс забезпечує зв'язок між користувачем і комп'ютером – він дозволяє досягати поставлених цілей, успішно знаходити рішення поставленого завдання. Взаємодія – обмін діями і реакціями на ці дії між комп'ютером і користувачем.

Мета створення ергономічного інтерфейсу полягає в тому, щоб відобразити інформацію настільки ефективно наскільки це можливо для людського сприйняття і структурувати відображення на дисплеї таким чином, щоб привернути увагу до найбільш важливих одиниць інформації. Основна ж мета полягає в тому, щоб мінімізувати загальну інформацію на екрані і представити тільки те, що є необхідним для користувача.

Існує ряд основних принципів побудови інтерфейсів:

• Принцип угруповання – згідно з цим правилом, екран програми повинен бути розбитий на ясно окреслені блоки елементів, може бути, навіть з заголовком для кожного блоку.

• Гаманець Міллера – ємність пам'яті обмежена сім'ю цифрами. Необхідне відповідне групувати суті в програмі (пункти меню,

закладки, опції на цих закладках і т. П.) Бажано з урахуванням цього правила – тобто не більше семи в групі, в крайньому випадку – дев'яти.

• Бритва Оккама або KISS:

- будь-яке завдання має вирішуватися мінімальним числом дій;
- логіка цих дій повинна бути очевидною для користувача;
- руху курсору і навіть очей користувача повинні бути оптимізовані.

• Видимість відображає корисність – винести найважливішу інформацію і елементи управління на перший план і зробити їх доступними користувачеві, а менш важливу – перемістити, наприклад, в меню.

Для розробників та користувачів ефективний інтерфейс має свої переваги.

Для розробників програмного забезпечення:

- · покращене представлення продукту;
- · підвищений попит на продукт;
- · більш короткий час навчання;
- · більш низькі вимоги до сервісу після продажу;
- · тверда основа, на якій відбувається розвиток серійного виробництва продукту;
- · зниження ризику помилкових дій і нещасних випадків;
- · скорочення кількості документації.

Для користувачів:

- · більш короткий час навчання;
- · підвищена загальна застосовність навичок;
- · зросла автономія при використанні системи;
- · зниження кількості часу, необхідного для виконання завдання;
- · зменшення кількості помилок;
- · зростання задоволення від роботи.

Для розробки зручного інтерфейсу на етапі його проектування складається діаграма прецедентів – Use Case.

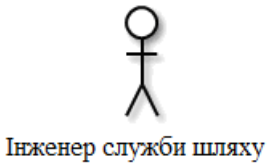
Діаграма варіантів використання (сценаріїв поведінки, прецедентів) є вихідним концептуальним поданням системи в процесі її проектування і розробки. Дана діаграма складається з акторів,

варіантів використання і відносин між ними. При побудові діаграми можуть використовуватися також загальні елементи нотації: примітки і механізми розширення.

Суть даної діаграми складається в наступному: проектована система представляється у вигляді безлічі акторів, що взаємодіють з системою за допомогою так званих варіантів використання. При цьому актором (дійовою особою) називається будь-який об'єкт, суб'єкт або система, що взаємодіє з моделюється системою ззовні. У свою чергу варіант використання – це специфікація сервісів (функцій), які система надає актору. Іншими словами, кожен варіант використання визначає деякий набір дій, що здійснюються системою при взаємодії з актором. При цьому в моделі ніяк не відбивається те, яким чином буде реалізовано цей набір дій.

У структурному підході аналогом діаграми варіантів використання є діаграми IDEF0 і DFD, варіантів використання - роботи (IDEF0) і процеси (DFD), акторів - зовнішні сутності (DFD).

Згідно нотацій UML ***актора*** графічно можна відобразити трьома способами (рис. 2.5.):



а) «дротяний чоловічок»

б) клас с текстовим стереотипом «actor»

в) довільна іконка

Рис. 2.5. Приклади представленняя акторів

Варіант використання позначається на діаграмі еліпсом, усередині якого міститься його опис, що означає виконання будь-якої операції або дії (рис. 2.6.).

Варіант використання, що ініціалізується за запитом користувача, є закінченою послідовність дій. Це означає, що після того, як система закінчить обробку запиту актора, вона повинна повернутися в стан, в якому готова до виконання наступних запитів.

Варіанти використання можуть включати в себе опис особливостей способів реалізації сервісу і різних виняткових ситуацій, таких як коректна обробка помилок системи.

а) б)

Рис. 2.6. Варіанти використання:
а) приклади варіантів використанняя;
б) приклад використання примітки

Примітки призначені для включення в діаграму довільної текстової інформації, що має безпосереднє відношення до контексту розроблюваної системи. В якості такої інформації можуть бути коментарі розробника і обмеження. Графічно примітки відображаються прямокутником із загнутим верхнім правим куточком, всередині якого міститься текст примітки. Лінія, що з'єднує примітка і елемент діаграми, називається якорем (фіксацією).

Зв'язки між акторами і варіантами відображаються з використанням відносин чотирьох видів, представлених у таблиці 2.1 та на рис. 2.7.-2.9.

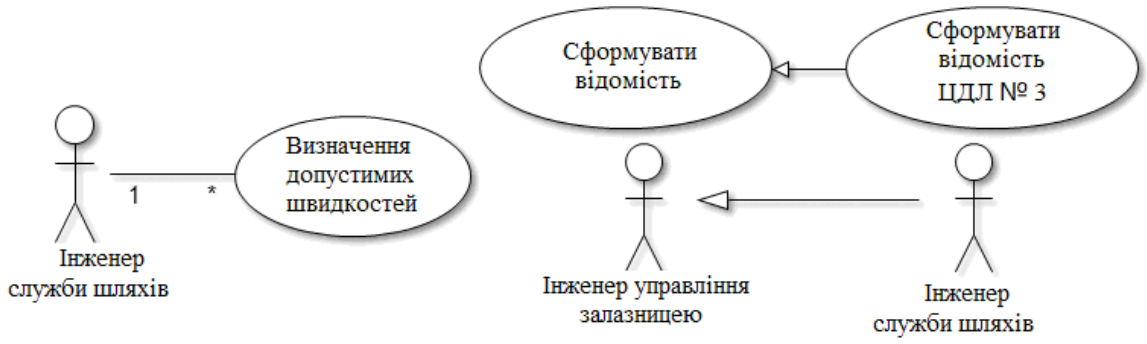
Таблиця 2.1.

Типи зв’язків діаграми Use Case

Тип	**Представ.**	**Призначення**
асоціації	———	Асоціації служить для позначення взаємодії актора з варіантом використання. Асоціація може відображатися у вигляді односпрямованої або двобічної стрілки, яка показує напрямок потоків інформації або сигналів.
узагальнення	——▷	Служить для вказівки того, що деяка сутність **A** може бути узагальнена до суті **B**. В цьому випадку сутність **A** буде спеціалізацією суті **B**.

Продовження таблиці 2.1.

Тип	**Представ.**	**Призначення**
		На діаграмі даний вид відносини можна відображати тільки між однотипними сутностями (між двома варіантами використання або двома акторами від потомка до батька).
включення	«include» ---------▶	Відношення включення вказує, що деяка задана поведінка одного варіанту використання обов'язково включається в якості складового компонента в послідовність поведінки іншого варіанту використання.
розширення	«extend» ---------▶	Відношення розширення визначає потенційну можливість включення поведінки одного варіанту використання до складу іншого. т. е. дочірній варіант використання може як викликатися, так і не викликатися батьківським. Стрілка розширення повинна бути спрямована від включається варіанту до базового і позначена стереотипом «extend» (англ. Розширює).



а) б)

Рис. 2.7. Приклади зв’язків.
а) асоціації
б) узагальнення

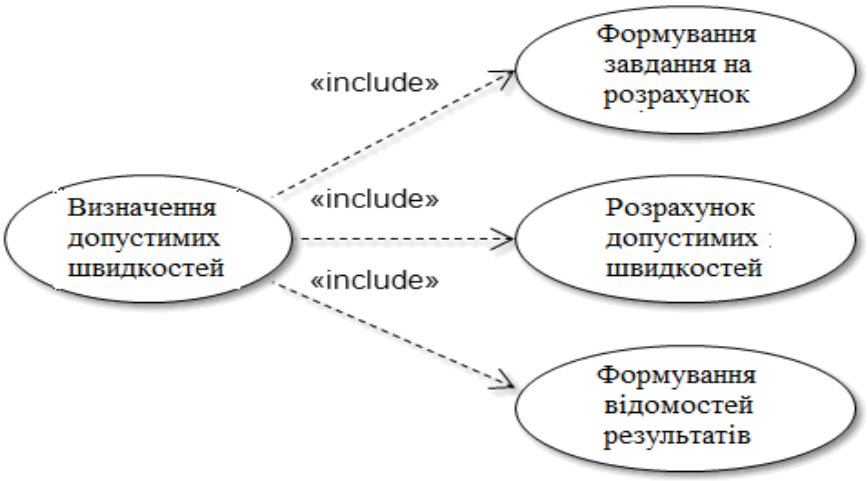


Рис. 2.8. Приклад включення

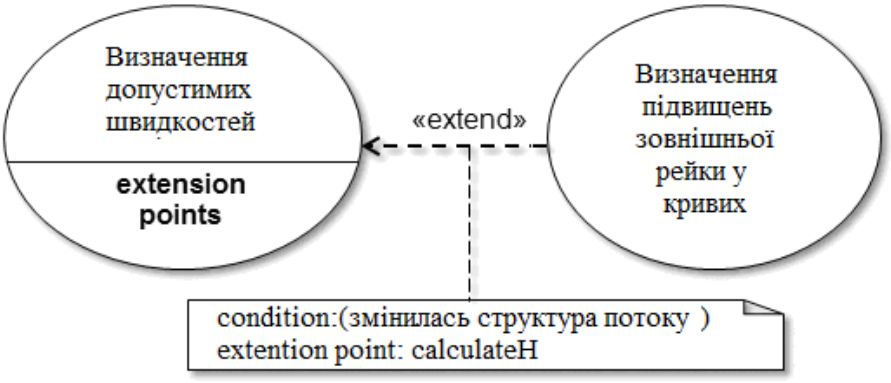


Рис. 2.9. Приклад росширення

З огляду на те, що допустима швидкість в кривих ділянках колії залежить від підвищення зовнішньої рейки, перед визначенням допустимих швидкостей може знадобитися визначення та встановлення нових підвищень, які в свою чергу залежать від структури потоку потягів.

Варіанти використання, які розширюють базовий, підключаються (активуються при його виконанні) через так звані точки розширення (англ. Extension points). Кожна точка розширення маркується міткою (рис. 2.10. – calculateH) і умовою (англ. Condition) активації. Зазвичай перелік точок розширення вказується в базовому варіанті використання нижче горизонтальної лінії.

Приклад діаграми прецедентів для завдання оформлення замовлення в бібліотеці наведено на рис. 2.10.

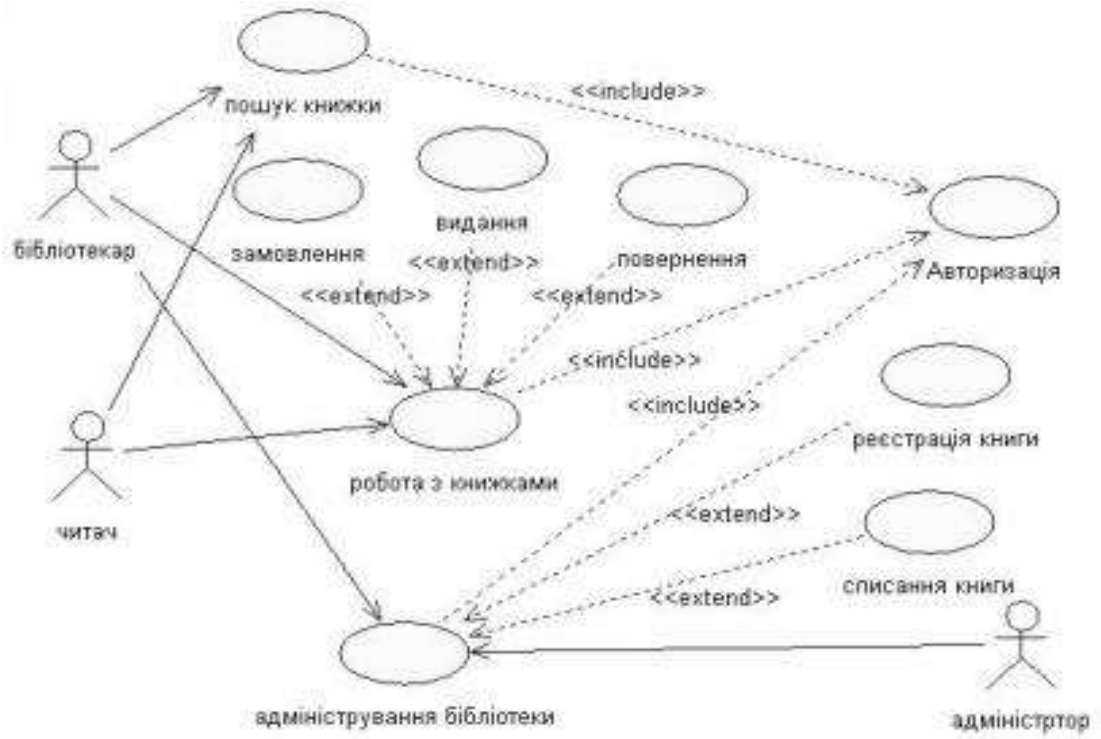


Рис. 2.10. Приклад діаграми прецедентів

Для представлення процесу, який реалізує додаток будується діаграми станів (рис. 2.11), які зображають всі можливі стани, в яких може знаходитися конкретний об'єкт, а також зміни стану об'єкту, що відбуваються в результаті впливу деяких подій на цей об'єкт.

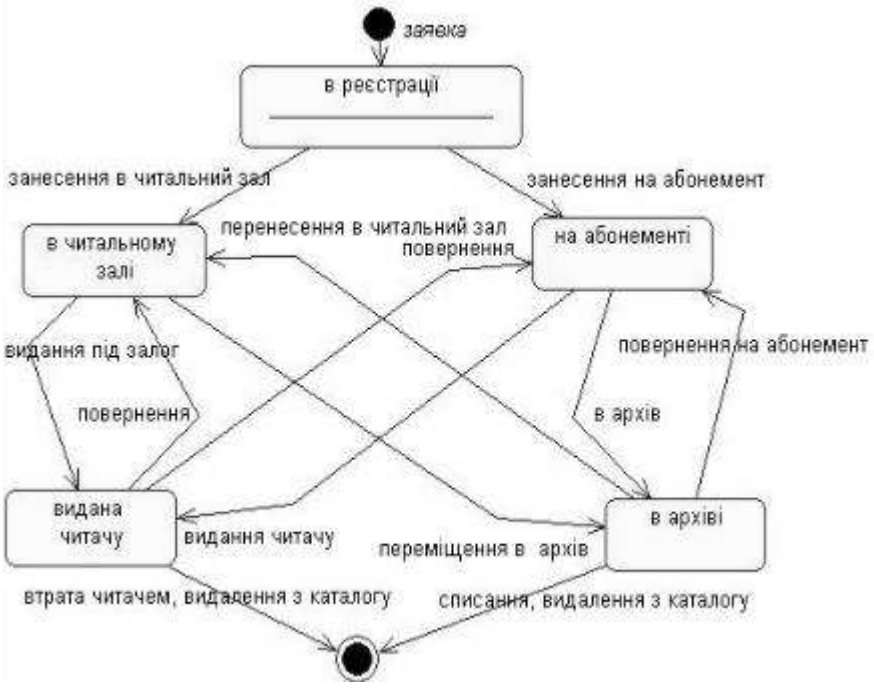


Рис. 2.11. Приклад діаграми станів

Контрольні питання

1. Які основні цілі з точки зору ергономіки необхідно досягти в результаті проектування програмного продукту?
2. Що таке програмній модуль?
3. Які переваги має модульна структура програми?
4. Як схематично представляється структура програми?
5. Що таке діаграма пакетів?
6. Що таке атрибутний метод проектування?
7. Перерахуйте основні етапи ADD.
8. Що таке методи ATAM, CBAM?
9. Що називається архітектурою програмного комплексу? Яка різниця між архітектурою та структурою ПЗ?
10. З яких основних компонентів складається архітектура програмного комплексу? Наведіть приклад.
11. Які виділяють основні типи архітектури?
12. Яка структурна частина ПЗ забезпечує взаємодію людини та системи?
13. Які основні задачі вирішує інтерфейс користувача?
14. Які основні вимоги висуваються до розробки користувацького інтерфейсу?
15. Які переваги дає вдалий інтерфейс для: а) розробників; б) користувачів?
16. За допомогою якої діаграми можна представити взаємодію користувача з програмним продуктом?
17. Які компоненти входять до складу діаграми Use Case?
18. Які типи зв’язків використовуються у діаграмі Use Case?
19. Наведіть приклад діаграми Use Case.
20. Що таке діаграма станів?
21. Наведіть приклад діаграми станів.

Лекція 3 Параметри, які необхідно враховувати при розробці інтерфейсу

3.1. Етапи життєвого циклу процесу розробки програмного інтерфейсу користувача (ПІ)

При аналізі проблеми створення сучасної системи «людина-машина» можна виділити у вигляді окремих аспектів наступні завдання:

- технічне проектування;
- інженерно-психологічне проектування;
- художнє проектування.

Таке проектування по своїй суті засновано на обліку різних людських потреб. Навіть при проектуванні «чисто» технічної системи, що діє автоматично, (досить незалежно від людини) доводиться враховувати, якою мірою вона буде задовольняти людину як засіб його праці за вартістю і багатьма іншими критеріями. При створенні ж систем «людина-машина», в яких людина виступає як компонент системи (зазвичай провідний), обсяг і широта вимог, що випливають з потреб людини, істотно зростає і необхідно врахувати всю цю складну сукупність вимог людини, яка керує системою, в поєднанні з вимогами технічного, економічного та інших порядків, можливо в наш час тільки при посередництві методів системотехніки.

Ергономіка систем в «людина-машина» включається в процеси розробки та тестування програмного продукту як частина системи якості. Розробка користувальницького інтерфейсу(ПІ), який значною мірою визначає ергономіку програмного продукту, ведеться паралельно дизайну програмного продукту в цілому і в основному передує його імплементації. Процес розробки ПІ розбивається на етапи життєвого циклу:

- Аналіз трудової діяльності користувача, об'єднання бізнес-функцій в ролі.
- Побудова користувальницької моделі даних, прив'язка об'єктів до ролей і формування робочих місць.
- Формулювання вимог до роботи користувача і вибір показників оцінки призначеного для користувача інтерфейсу.

- Розробка узагальненого сценарію взаємодії користувача з програмним модулем (функціональної моделі) і його попередня оцінка користувачами і Замовником.
- Коригування і деталізація сценарію взаємодії, вибір і доповнення стандарту (керівництва) для побудови прототипу.
- Розробка макетів і прототипів ПІ і їх оцінка в діловій грі, вибір остаточного варіанту.
- Імплементація ПІ в коді, створення тестової версії.
- Розробка засобів підтримки користувача (словники, підказки, повідомлення, допомогу і ін.) І їх вбудовування в програмний код.
- Usability тестування тестової версії ПІ по набору раннє певних показників.
- Підготовка документації для користувачів і розробка програми навчання.

3.2. Ергономічні цілі і показники якості програмного продукту

Програмний додаток розробляється для забезпечення роботи користувача, тобто для того щоб він за допомогою комп'ютерної програми швидше і якісніше вирішував свої виробничі завдання.

З точки зору ергономіки, найважливіше в програмі – створити такий призначений для користувача інтерфейс, який зробить роботу ефективною і продуктивною, а також забезпечить задоволеність користувача від роботи з програмою.

Ефективність роботи означає забезпечення точності, функціональної повноти і завершеності при виконанні виробничих завдань на робочому місці користувача. Створення ПІ має бути націлене на показники ефективності:

- *точність роботи*

визначається тим, якою мірою вироблений користувачем продукт (результат роботи), відповідає пред'явленим до нього вимогам. Показник точності включає відсоток помилок, які зробив користувач: число помилок набору, варіанти помилкових шляхів або відгалужень, число неправильних звернень до даних, запитів тощо.

- *функціональна повнота*

відображає ступінь використання первинних і оброблених даних, списку необхідних процедур обробки або звітів, число пропущених технологічних операцій або етапів при виконанні поставленого користувачеві завдання. Цей показник може визначатися через відсоток застосування окремих функцій в РМ.

- *завершеність роботи*

описує ступінь виконання виробничого завдання середнім користувачем за певний термін або період, частку (або довжину черги) незадоволених (необроблених) заявок, відсоток продукції, що знаходиться на проміжній стадії готовності, а також число користувачів, які виконали завдання у фіксований термін.

Послідовність дій і набір інструментальних засобів користувача в ПІ повинні бути підпорядковані технологічним процесам виконання виробничого завдання.

Не треба боятися складності системи, треба уникати такого інтерфейсу, який не відповідає алгоритму вирішення користувальницьких задач.

3.3. Основні характеристики, що враховуються при розробці ПІ користувача

Необхідно ретельно продумати і усвідомити сценарій взаємодії програми з користувачем, привівши його до оптимальної (щодо розглянутих показників) системі виконання завдань, і реалізувати ПІ відповідно до цієї системою.

Для того, щоб розібратися в технології вирішення завдань користувача, розробнику необхідно з'ясувати наступні моменти (досліджуючи діяльність користувача):

- Яка інформація необхідна користувачеві для вирішення завдання?
- Яку інформацію користувач може ігнорувати (не враховувати)?
- Спільно з користувачем розділити всю інформацію на сигнальну, що відображається, що редагується, пошукову і результуючу.

- Які рішення користувачеві необхідно приймати в процесі роботи з програмою?
- Чи може користувач здійснювати кілька різних дій (вирішувати кілька завдань) одночасно?
- Які типові операції використовує користувач при вирішенні задачі?
- Що станеться, якщо користувач буде діяти не по запропонованому Вами алгоритму, пропускаючи ті чи інші кроки або обходячи їх?

Продуктивність роботи відображає обсяг витрачених ресурсів при виконанні завдання, як обчислювальних, так і психофізіологічних. Яка інформація необхідна користувачеві для вирішення завдання?

- Яку інформацію користувач може ігнорувати (не враховувати)?
- Спільно з користувачем розділити всю інформацію на сигнальну, що відображається, що редагується, пошукову і результуючу.
- Які рішення користувачеві необхідно приймати в процесі роботи з програмою?
- Чи може користувач здійснювати кілька різних дій (вирішувати кілька завдань) одночасно?
- Які типові операції використовує користувач при вирішенні задачі?
- Що станеться, якщо користувач буде діяти не по запропонованому Вами алгоритму, пропускаючи ті чи інші кроки або обходячи їх?

Продуктивність роботи відображає обсяг витрачених ресурсів при виконанні завдання, як обчислювальних, так і психофізіологічних.

Дизайн ПІ повинен забезпечувати мінімізацію зусиль користувача при виконанні роботи і приводити до:

- скорочення тривалості операцій читання, редагування і пошуку інформації,
- зменшення часу навігації і вибору команди,
- підвищенню загальної продуктивності користувача, що полягає в обсязі оброблених даних за певний період часу.
- збільшення тривалості стійкої роботи користувача і ін.

Скорочення невиробничих витрат і зусиль користувача – важлива складова якості програмного забезпечення.

Для оцінки продуктивності використовуються відповідні показники, що перевіряються фахівцями з ергономіки в процесі usability тестування робочого прототипу.

Формування таких показників відбувається в процесі визначення вимог до ПІ при вивченні таких питань:

Що від користувача потрібно в першу чергу?

- Скільки інформації, що вимагає обробки, надходить користувачеві за період часу?
- Які вимоги до точності і швидкості введення інформації?
- На які операції користувач витрачає найбільше часу?
- Чим ми можемо полегшити роботу користувача при вирішенні типових завдань?

3.4. Вимоги до зручності і комфортності інтерфейсу

Задоволеність користувача від роботи тісно пов'язана з комфортністю його взаємодії з додатком, і сприяє збереженню професійних кадрів на підприємстві Замовника за рахунок привабливості роботи на даному робочому місці.

Вимоги до зручності і комфортності інтерфейсу зростають зі збільшенням складності робіт і відповідальності користувача за кінцевий результат. Висока задоволеність від роботи досягається в разі:

- Прозорою для користувача навігації і цільової орієнтації в програмі. Головне, щоб було зрозуміло, куди йдемо, і яку операцію програма після цього кроку зробить.
- Ясності й чіткості розуміння користувачем текстів і значення ікон. У програмі повинні бути ті слова і графічні образи, які користувач знає або повинен знати за характером його роботи або займаної посади.
- Швидкості навчання при роботі з програмою, для чого необхідно використовувати переважно стандартні елементи взаємодії, їх традиційне або загальноприйняте їх розташування.

- Наявності допоміжних засобів підтримки користувача (пошукових, довідкових, нормативних), в тому числі і для прийняття рішення в невизначеній ситуації (введення за замовчуванням, обхід «зависання» процесів і ін.).

При розробці зручного та ефективного інтерфейсу враховується профіль користувача – набір дозволених підказок, операцій і класів даних, доступ до яких необхідний користувачам для виконання певних робочих обов'язків. Прийнято розрізняти три типи користувачів:

- добре підготовлених для роботи з програмним додатком. Для таких користувачів додаткова інформація буває зайвою і навіть заважає;
- середньо рівня. Для таких користувачів передбачаються підказки, які, наприклад, з’являються при затримці курсору на певному елементі інтерфейсу;
- новачків. Для таких користувачів доступні як спливаючі підказки так і розгорнута документація стосовно необхідних операцій до якої можна дістатись, наприклад, при натисканні клавіші F1.

Для оцінки необхідного рівня зручності інтерфейсу також використовуються спеціальні опитувальники, формуляри, чек-листи, однак до даної роботи краще залучати фахівців з ергономіки.

Зручний інтерфейс допомагає користувачеві впоратися з втомою і напругою при роботі в умовах високої відповідальності за результат.

3.5. Проблеми розробки прототипу інтерфейсу

Проблеми, що виникають на етапі розробки прототипу інтерфейсу і варіанти їх вирішення

1. Облік особливостей пристроїв введення / виводу інформації, використовуваних користувачем, наприклад:

- розмір екрану монітора;
- розширення екрану;
- палітра кольорів;
- характеристики звуку (якість відтворення мови) і відеокарти (швидкість виведення при анімації);

- вид миші (з роликом або без);
- тип клавіатури ("пряма", "коса");
- необхідність додаткового обладнання (штрих-декодера, світлового пера сенсорного екрану і ін.).

2. Специфіка інтерактивних елементів, пов'язана з вибором платформи, стандартних бібліотек:

- програмна організація введення / виведення інформації;
- зміна і створення нових елементів форм (контролів);
- придбання нестандартних бібліотек у інших фірм.

3. Вибір технології та методів ведення діалогу програми з користувачем:

- ступінь активності користувача при взаємодії (автоматичний режим або перехоплення управління програмою на себе, забезпечення доступу до всіх засобів інтерфейсу незалежно від дій користувача);
- ступінь врахування ситуації (контекстні підказки, меню подальших подій або об'єктів, запам'ятовування типових шляхів діалогу);
- відповідність очікуванням користувача (прогноз, попередня обробка, предформатування);
- стійкість, терпимість до помилок користувача шляхом виправлення типових помилок;
- дублювання вручну окремих функцій системи і додаткові контрольні процедури роботи окремих режимів;
- настройка ПІ на різний рівень підготовки користувача (образність або метафоричність предметної області на противагу скорочень і гарячих клавішах);
- ступінь адаптивності ПІ під переваги користувача (зміна способу і порядку відображення, перекомпонування екрану, вибір окремих характеристик (стилю) та ін.);
- настройка ПІ на специфіку завдання (новий формат даних, зміна набору об'єктів, доповнення атрибутів об'єктів).

4. Розміщення інформації і керуючих елементів в поле екрану, в вікні. При композиції екрану необхідно враховувати обмежені розміри простору екрану, в зв'язку з чим виникає задача оптимального розташування максимально можливого обсягу інформації шляхом:

- логічної ув'язкою даних в залежності від алгоритму роботи користувача, а не орієнтацією на структуру і послідовність фізичних таблиць даних;
- визначення рівня "детальності - узагальненості" виведення інформації (знаходження компромісу між бажанням вивести багато записів одночасно і / або відразу побачити детальну інформацію по кожній з них);
- виділення важливої інформації на екрані;
- чіткого визначення основних і допоміжних блоків інформації;
- визначення статичних полів на екрані, а також полів, де інформація періодично змінюється;
- уникнення перекриваються вікон на екрані;
- застосування принципів гармонії при компонуванні екрану (симетрія, балансу мас, дотримання пропорцій, поєднання кольорів).

5. Формування зворотного зв'язку між користувачем і додатком:

- показ актуального стану системи, режиму роботи системи (автономного, штатного, захищеного і ін.) і режиму взаємодії (наприклад, відображення, редагування або пошук даних);
- виведення окремих, важливих для робочої операції даних і показників;
- відображення дій користувача (натискання клавіш, запуск процесу, динаміка виконання процесу, отримання очікуваного і іншого результату);
- ясність і інформативність повідомлень системи. При роботі з автоматизованими слідкуючими системами у користувача може спостерігатись втрата уваги, тому, у випадку настання особливих ситуацій доцільно використовувати звукові сигнали (наприклад, у системах стеження за температурою у зоні реактора). Для розмежування вхідної та вихідної інформації можна використовувати різні кольори або яскравість (особливо це доцільно коли швидкість виконання дії менше 2 с.). Коли виконання операції вимагає часу більш ніж 20 с., у користувача може з’явитись відчуття, що програма не працює, у таких випадках доцільно виводити на екран проміжну інформацію або відзначати робочий стан біжучим рядком.

6. Проектування панелей меню та інструментів (toolbars) і вибір пунктів в них:

- логічна і смислова угруповання пунктів;
- фіксована позиція панелей на екрані;
- обмеження на ширину списку виборів і кроків (глибини) меню;
- використання звичних назв, широко поширених ікон-піктограм, традиційних ікон-символів і акуратне введення скорочень;
- розміщення найбільш часто використовуваних пунктів (зазвичай на початку списку).

7. Розробка засобів орієнтації і навігації:

- легкість визначення свого місцезнаходження і вказівка напрямку курсування;
- зручний перехід від узагальненого погляду до конкретних деталей (варіювання ступеня деталізації аналізованих об'єктів);
- швидкий пошук в списку або таблиці;
- вказівка на додатково існуючу інформацію і спосіб її отримання;
- використання коштів перегортання і прокрутки.

8. Створення форм для введення даних:

- використання одного або декількох механізмів введення в рамках режиму (клавіатура, миша, штрих-декодер, світлове перо, ін.);
- визначення способів введення даних (таблиці, списки, проста форма, меню та ін.);
- мінімізація обсягу введення;
- виділення редагованих обов'язкових і необов'язкових, а також нередагуємих полів;
- використання механізмів швидкого введення (за замовчуванням, скорочення, з продовженням та ін.);
- Виділення введеної або відредагованого інформації.

3.6. Принципи реалізації призначеного для користувача інтерфейсу

Стильова гнучкість – можливість використовувати різні інтерфейси з одним і тим же додатком, на практиці реалізується у

вигляді набору "skins", для web-інтерфейсів – за допомогою таблиці стилів, в тому числі можливість у виборі користувачем власних установок ПІ (колір, ікони, підказки тощо .).

Спільне нарощування функціональності – можливість розвивати додаток без руйнування (тобто залишаючись в рамках) існуючого інтерфейсу.

Масштабованість – можливість легко налаштовувати і розширювати як інтерфейс, так і сам додаток при збільшенні числа користувачів, робочих місць, обсягу і характеристик даних.

Адаптивність до дій користувача – додаток повинен допускати можливість введення даних і команд безліччю різних способів (клавіатура, миша, інші пристрої) і багато варіативність доступу до прикладних функцій (ікони, «гарячі клавіші», меню ...), крім того програма повинна враховувати можливість переходу і повернення від вікна до вікна, від режиму до режиму, і правильно обробляти такі ситуації.

Незалежність в ресурсах – для створення призначеного для користувача інтерфейсу повинні надаватися окремі ресурси, спрямовані на зберігання і обробку даних, необхідних для підтримки користувача (словники, контекстно-залежні списки, набори даних за замовчуванням або за останнім запитом, історії запитів та ін.)

Можливість перенесення – при переході на іншу апаратну (програмну) платформу, повинен здійснюється автоматично перенесення і призначеного для користувача інтерфейсу, і кінцевого додатки.

3.7. Типи інтерфейсу

Інтерфейс в основному виконує дві функції:

- введення інформації користувачем;
- відображення виведення результатів.

Існує два основних типи призначеного для користувача інтерфейсу:

а) текстовий;

б) графічний.

Програмне забезпечення в різних доменах може вимагати різного стилю свого призначеного для користувача інтерфейсу,

наприклад, для калькулятора потрібно тільки невелика область для відображення числових даних, але велика область для команд, web-сторінці потрібні форми, посилання, вкладки і т. д.

Графічний інтерфейс користувача є найбільш поширеним типом інтерфейсу, доступним сьогодні. Це дуже зручно для користувача, тому що він використовує малюнки, графіку і значки – тому його називають «графічним».

Він також відомий як інтерфейс WIMP, тому що він використовує:

Windows – прямокутна область на екрані, де запускаються часто використовувані додатки. Інтерфейс складної системи може складатись з декількох вікон – екранних форм (рис. 3.1.).

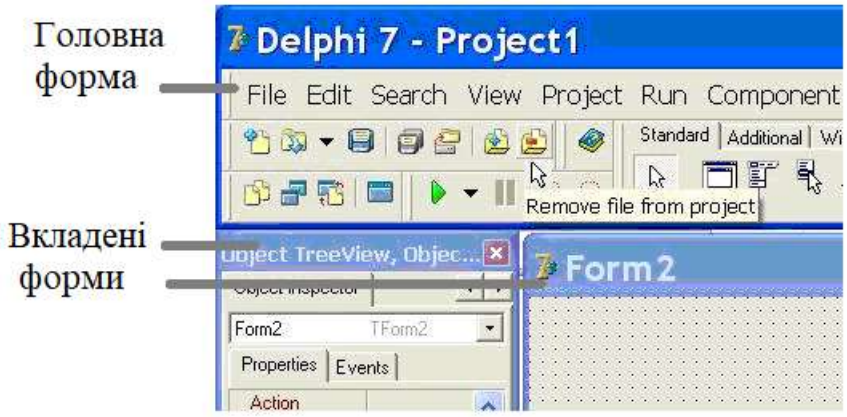


Рис. 3.1. Приклад фрагменту представлення вкладених екранних форм

Icons – зображення або символ, який використовується для представлення програмного додатка або апаратного пристрою.

Піктограми роблять інтерфейс більш привабливим в візуальному плані і, за певних умов, можуть сприяти більшій зрозумілості. Однак є і недоліки піктограм. Наприклад, як в операційній системі Windows вже використовуються підказки для пояснення значення піктограм. Якщо ви наводите курсор на якусь піктограму, з'являється невелике вікно з текстом, в якому дається її опис.

Використання піктограм замість слів цілком підходить для того, щоб приховати або зашифрувати якусь інформацію від сторонніх очей.

Піктограма невеликого розміру займає менше екранного простору, ніж напис і сприймається значно швидше. Для ефективного використання піктограм їх представлення повинно відповідати прийнятим, зрозумілим асоціаціям для користувача, який буде користуватись конкретним програмним продуктом.

У багатьох випадках іконки, створені з хорошим знанням аудиторії, успішно замінюють текст в інтерфейсі і мають такі переваги:

- потенціал багатомовного сприйняття – зображення з розпізнаваними фізичними об'єктами можуть бути зрозумілі людьми, що говорять на різних мовах, тому іконки можуть зробити інтерфейс більш гнучким;
- збільшення швидкості взаємодії – фізично, переважна більшість користувачів сприймає зображення швидше, ніж слова;
- більш високий рівень запам'ятовування призначеного для користувача інтерфейсу – у зображень більше потенціал залишитися довгостроковій пам'яті, що може мати вирішальний вплив на запам'ятовування моделей взаємодії;
- економія місця в макеті – в разі, якщо зображення добре розпізнається і не викликає подвійного значення, воно може замінити текст і зберегти дорогоцінний простір для інших елементів макета, це особливо корисно для мобільних інтерфейсів, обмежених простором екрана;

У багатьох випадках іконки, створені з хорошим знанням аудиторії, успішно замінюють текст в інтерфейсі (рис. 3.2.), розглянемо всі переваги нижче:

Список характеристик для ефективних іконок. Вони повинні бути такими якостями:

- ясність (сенс зрозумілий цільовій аудиторії);
- значимість (інформативне значення передається);
- розпізнання (візуальний символ може бути правильно трактуватись користувачами);
- простота (зображення не перевантажене несуттєвими графічними елементами для швидкого сприйняття);
- масштабованість і гнучкість (іконка зберігає чіткість і цілісність в різних розмірах і дозволах);

- відповідає стилістиці макета.

Рис. 3.2 Приклад піктограм текстового редактора

Menu – список опцій, з яких користувач може вибрати те, що йому потрібно (рис. 3.3.).

Розрізняють такі типи меню:

- за виконанням:
  - текстове;
  - графічне.
- за функціями:
  - головне меню програми;
  - спливаюче меню;
  - контекстне меню;
  - системне меню.

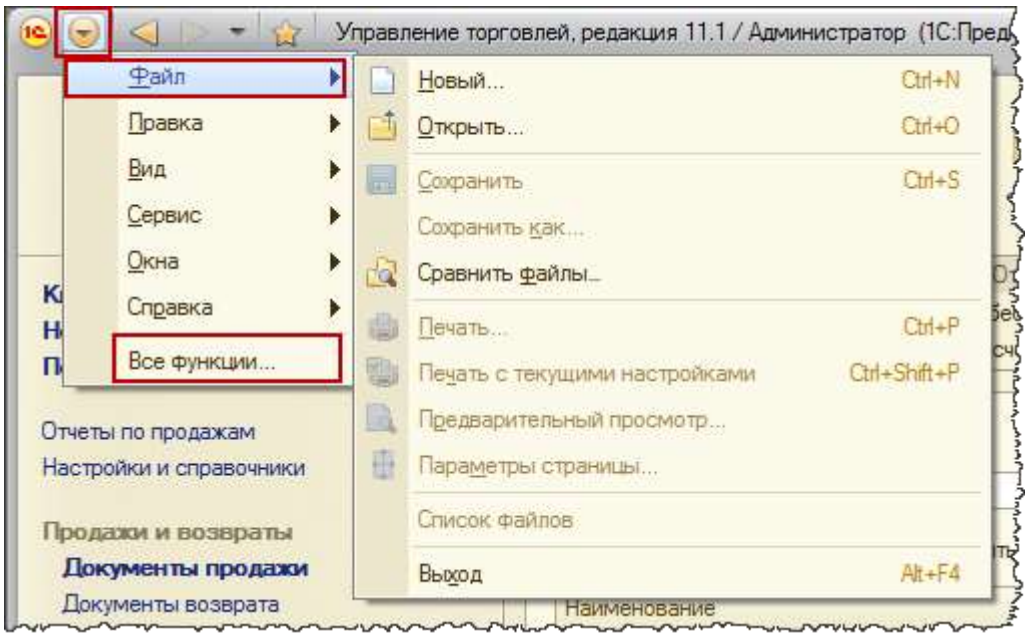


Рис. 3.3. Приклади меню різного типу

Pointer – символ (показчик), такий як стрілка, який переміщається по екрану, коли користувач переміщує миша. Це допомагає користувачеві вибирати об'єкти.

Форма покажчика може змінюватися в залежності від об'єкта і режиму взаємодії з ним. Наприклад, в графічних редакторах курсор має форму поточного обраного інструменту. Також сучасні операційні системи та графічні середовища підтримують кілька стандартних видів курсору:

 Css cursor default Курсор за замовчуванням – стрілка. Передбачається, що об'єкт під таким курсором самостійно повідомляє користувачеві своїм виглядом, чи можна з ним взаємодіяти;

 Css cursor pointer Курсор-рука – для позначення гіперпосилань, щоб повідомити користувачеві про те, що гіперпосилання працює;

 Css cursor crosshair Хрестик – використовується перш за все для графічного виділення;

 Css cursor text Текстовое виділення – сигналізує про те, що в це поле можна вводити текст, а також для нього діють виділення і контекстні меню, характерні для полів текстового введення;

 Css cursor move Курсор переміщення - сигналізує про те, що для обраного об'єкта доступна функція перетягування;

Курсори зміни розмірів:

 Monocursors з верхнього нижнього країв,

 Monocursors з лівого і правого краю, а також з кутів:

 Monocursors лівого верхнього і правого нижнього,

 Monocursors лівого нижнього і правого верхнього;

 Css cursor help Курсор режиму довідки – сигналізує, що для даного об'єкта після натискання кнопки миші доступна довідка, або є підказка;

 Monocursors Курсор фонового режиму - показує, що дія запущено, але видимих результатів може не бути. Такий Курсор для того, щоб користувач побачив, що команда прийнята і повторно натискати на об'єкт не потрібно. У деяких графічних середовищах поруч з курсором відображається рухається значок запущеного додатку;

 Css cursor wait Курсор очікування – повідомляє користувачеві, що в поточний момент взаємодія з елементом

неможливо через виконання програмою якихось операцій. Після закінчення виконання курсор повинен повернутися в початковий стан;

 курсор заборони дії – повідомляє, що якась агресивна дія (перетягування, натискання і т. д.) для даного елемента недоступно.

Зміна форми покажчика для конкретного елемента є також для веб-сторінок за допомогою властивостей CSS (Cascading Style Sheets – каскадні таблиці стилів) – формальна мова опису зовнішнього вигляду документа, написаного з використанням мови розмітки.

В операційних системах сімейства Windows курсори зберігаються в файлах з розширеннями .cur (для нерухомих версій) і .ani (анімовані курсори). Всі вони являють собою зображення розміром 32 × 32 пікселя і можуть мати кілька колірних варіацій для різної глибини кольору, встановленої в системі (True Color, HiColor, 256 кольорів, 16 кольорів, монохромний).

Контрольні питання

1. Перерахуйте основні етапи життєвого циклу процесу розробки програмного інтерфейсу.
2. Визначте основні завдання розробки користувацького інтерфейсу.
3. Яка основна ціль розробки ергономічного ПІ?
4. Визначте основні показники ефективності ПІ.
5. Які основні характеристики враховуються при розробці ПІ?
6. Що таке продуктивність роботи програмного продукту?
7. Що необхідно враховувати для підвищення продуктивності роботи програмного продукту?
8. Що таке профіль користувача? Які основні типи профілю користувача ви знаєте?
9. Які проблеми виникають при розробці ПІ?
10. Якими принципами необхідно керуватись при розробці ПІ?
11. Які виділяють основні типи інтерфейсу?
12. Що означає WIMP інтерфейс?
13. Що таке вікно або екранна форма інтерфейсу?
14. Що таке піктограми і які переваги має їх використання?
15. Для чого використовують меню інтерфейсу?

Лекція 4. Вимоги до процесів інтерфейсу та проектування і реалізація його компонентів

4.1. Вимоги до вводу / виведення даних

Як було зазначено раніше, основне призначення користувацького інтерфейсу – це процедури вводу / виведення інформації яка може бути у графічному вигляді або у вигляді даних (рис. 4.1.). Для успішного та ефективного виконання цих процедур розробнику необхідно чітко визначити і правильно описати у програмі наступні основні характеристики даних:

- тип даних: простий чи складний, текстовий чи числовий;
- обмеження;
- розмір поля для їх представлення;
- формат;
- параметри шрифту (тип, колір і т.д.).

Наступним кроком є визначення способу вводу даних:

- з клавіатури;
- за допомогою ручного маніпулятора «миші»;
- отримання з датчиків і т ін..

Виведення результатів може виконуватись:

- на екран;
- на друк (отримання жорсткої копії);
- на магнітні носії (файл);
- на датчики та ін..

Також необхідно передбачити та визначити всі необхідні повідомлення для ведення успішного діалогу користувача з програмним додатком. Це можуть бути повідомлення:

- про стан пристроїв;
- про стан програми;
- про стан виконання певних операцій;
- про помилки виконання операцій;
- підказки та ін..

Для ефективного виконання завдань, для яких призначена програма, доцільно передбачити перевірку на коректність інформації, що вводиться. Тому процес вводу доцільно організувати у вигляді наступної циклічної структури:

Повторити:
запит на введення даних;
прийняти вхідну інформацію;
перевірити вхідну інформацію;
поки не буде введена правильна інформація.

Перейти до виконання наступних дій.

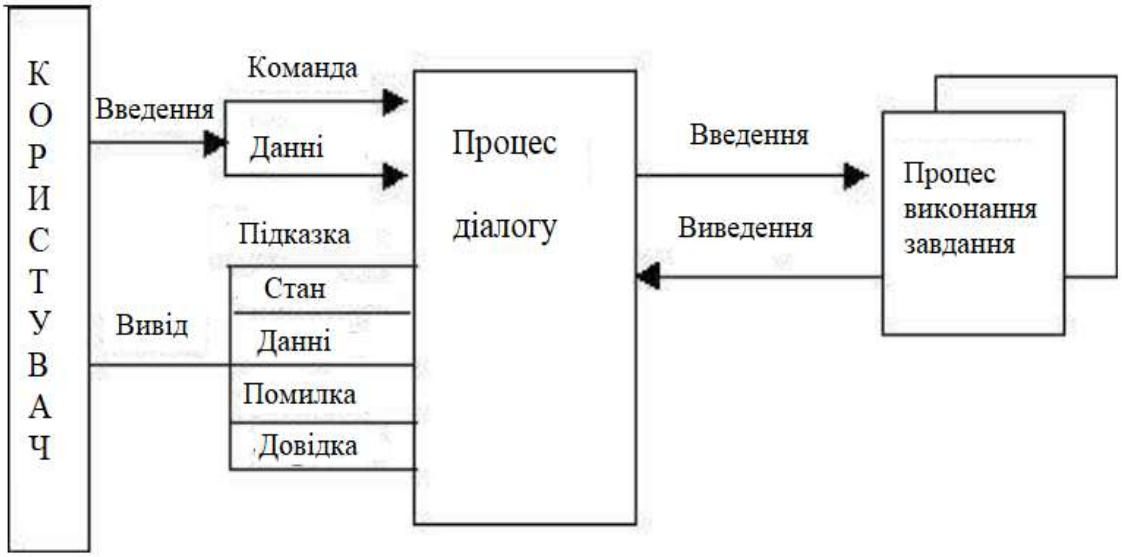


Рис. 4.1. Процес вводу/виводу та типи повідомлень

Інтерфейс має свій синтаксис і семантику. Синтаксис включає в себе типи компонентів, такі як текстовий, значок, кнопка і т. д., а зручність використання підсумовує семантику призначеного для користувача інтерфейсу. Якість призначеного для користувача інтерфейсу характеризується його зовнішнім виглядом (синтаксис) і зручністю використання (семантика).

Виділяють наступні основні компоненти на основі яких будується інтерфейс користувача:

1) екранні форми або вікна;
2) електронні таблиці та стандартні бланки;
3) діалог типу «питання-відповідь»;
4) меню
5) команди.

При проектуванні дизайну інтерфейсу необхідно відповісти на такі питання:

- *що* буде вводитись або виводитись (зміст повідомлень);
- *де* буде інформація вводитись або виводитись (розташування);
- *як* буде інформація вводитись або виводитись (спосіб, формат, колір та яскравість, вирівнювання, ехо-друк та ін..).

Розробку графічного інтерфейсу користувача можна робити на дві задачі:

1) дизайн (зовнішній вигляд);
2) програмну реалізацію.

Сьогодні,для створення привабливих інтерфейсів застосовуються об’єктно-орієнтований програмний інструментарій до якого відноситься і середовище Visual Studio. Середовище надає

розробнику бібліотеку візуальних компонентів для побудови багатофункціонального та багатокомпонентного інтерфейсу користувача. Для створення графічних додатків на мові C, C++ необхідні наступні компоненти: VC++ 20** v141 toolset (x86,x64); Windows 10 SDK (10.0.15063.0) for Desktop C++ x86 and x64; C++/CLI support. Перші два потрібні для створення будь-якого додатку, а третій – для графічного інтерфейсу.

4.2. Створення простого додатку Windows Forms Application.

Щоб почати роботу над новим додатком (проектом), треба:

1. У меню File вибрати команду New Project.

2. У вікні, New Project в списку Visual C ++ вибрати спочатку тип програми – CLR, а потім вид програми – Windows Forms Application.

3. У поле Name ввести ім'я проекту – profit і натиснути кнопку OK.

Робота над додатком починається із створення стартової форми – головного вікна програми. Спочатку потрібно встановити необхідні значення властивостей форми, потім помістити на форму необхідні компоненти (поля введення інформації, командні кнопки, поля відображення тексту і ін.).

Налаштування форми (а також компонентів) здійснюється шляхом зміни значень властивостей. Властивості об'єкта (форми, компонента) визначають його вигляд і поведінку. Наприклад, властивість StartPosition – визначає положення вікна в момент появи його на екрані.

Детальніше роботу з вікнами розглянемо далі.

Далі на формі встановлюються компоненти графічного інтерфейсу (рис. 4.2). Поля відображення тексту, командні кнопки, списки, перемикачі та інші елементи, що забезпечують взаємодію користувача з програмою, називають компонентами користувальницького інтерфейсу. Вони знаходяться у вікні Tollbox на вкладці Common Control.

Щоб на форму додати компонент, треба:

1. У палітрі компонентів (вікно Toolbox) розкрити вкладку Common Control. Встановити курсор на значку компонента.

2. Встановити покажчик миші в ту точку форми, в якій повинен бути лівий верхній кут компонента, і зробити клацання лівою кнопкою миші. Компонент з’явиться на формі. Кожному доданому компоненту середовище розробки привласнює ім'я, яке складається з назви компонента і його порядкового номера. Наприклад, перший доданий на форму компонент TextBox отримує ім'я textBox1, другий –

textBox2. Програміст шляхом зміни значення властивості Name може змінити ім'я компонента. Однак в простих програмах імена компонентів, як правило, не змінюють.

3. Виконати налаштування властивостей компонента. Щоб побачити і, якщо треба, змінити властивості компонента, потрібно цей компонент вибрати – клацнути лівою кнопкою миші на зображенні компонента в формі або вибрати його ім'я в списку, який знаходиться у верхній частині вікна Properties. У вікні Properties зробити необхідні зміни.

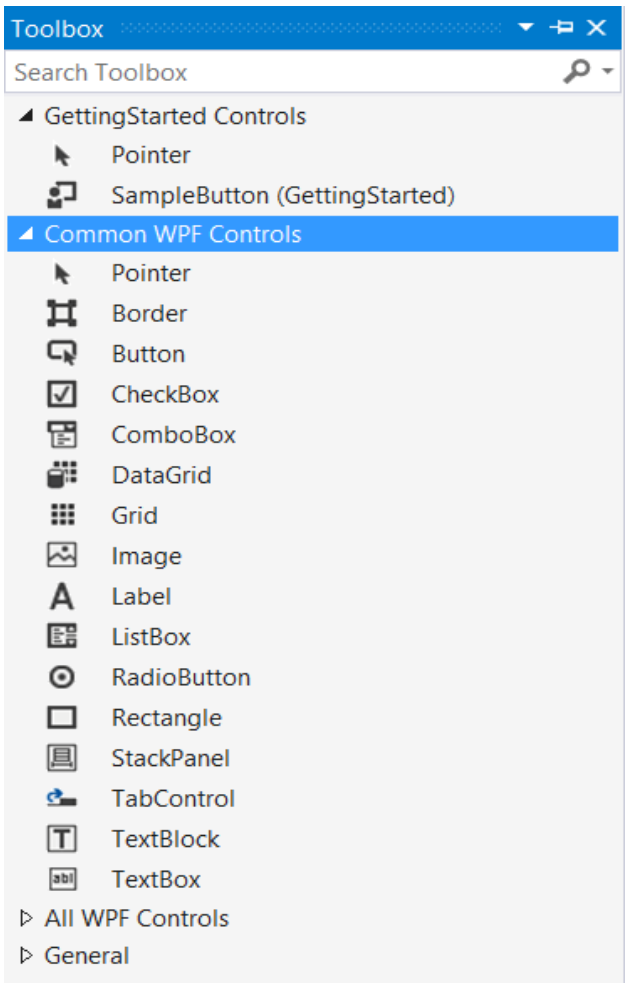


Рис. 4.2. Панель бібліотеки візуальних елементів ToolBox

4. Пов’язати компонент з певною подією (табл. 4.1). Подія (event) – це те, що відбувається під час роботи програми. Наприклад, клацання кнопкою миші - це подія Click, подвійне клацання мишею – подія DblClick.

Для того щоб створити функцію обробки події, потрібно на вкладці Evens вибрати подію (зробити клацання мишею на імені події), в поле значення властивості ввести ім'я функції обробки події і натиснути клавішу <Enter>. В результаті цих дій в модуль форми (h-файл) буде додана функція обробки події і стане доступним вікно

редактора коду, в якому можна набирати інструкції, що реалізують функцію обробки події.

Таблиця 4.1. Події

Подія	Опис
Click	Клацання кнопкої миші.
DoubleClick	Подвійне клацання кнопкої миші.
MouseDown	Натискання на кнопку миші.
MouseUp	Відпускання натиснутої кнопки миші.
MouseMove	Переміщення вказівника миші.
KeyPress, KeyDown	Натиснення клавіші на клавіатурі. Це події, які повторюються поки не буде відпущена натиснута клавіша (відбудеться подія KeyUp).
KeyUp	Відпускання натиснутої клавіші.
TextChanged	Ознака, яка визначає, чи відбулася зміна тексту (змінилось значення властивості Text).
Load	Завантаження формию
Paint	Подія відбувається при появі вікна на екрані на початку роботи програми, після появи частини вікна, яка, наприклад, була закрита іншим вікном.
Enter	Отримання фокусу елементом управління.
Leave	Втрата фокусу елементом управління.

4.3. Екранні форми або вікна

4.3.1. Понятя вікна та їх класифікація

Вікно або екранна форма – це прямокутна частина екрану або весь екран, через яку користувач спостерігає за окремими аспектами своєї взаємодії з програмним додатком.

Екранні форми можна класифікувати по ряду ознак.

1. За *характером зв'язку з таблицями* розрізняють зв'язані і незв'язані екранні форми. Якщо форма відображає будь-які дані з таблиць баз даних, вона називається пов'язаною (або приєднаною), в іншому випадку – незв'язаною.

За кількістю використовуваних таблиць виділяють одно табличну і багато табличні форми.

За характером підпорядкування окремих частин багато табличні форми класифікуються як прості, ієрархічні і синхронізовані. Прості багато табличні форми хоча й містять дані з різних таблиць, але не мають в своєму складі супідрядних частин.

2. По *виконанню функцій* розрізняють форми введення, виведення, керуючі, змішані. Призначення кожного виду цих форм ясно з їх назви.

3. За *розподілом даних* по екранах (сторінок) форми діляться на одно сторінкові і багатосторінкові; однією з різновидів багатосторінкових форм можна вважати форми з вкладками.

4. За *способом реалізації* екранні форми можуть бути спливаючими і не спливаючими. Спливаюча форма розташовується поверх інших відкритих форм, навіть якщо активною є інша форма. Спливаюча форма може бути немодального або модальної. Якщо спливаюча форма – модальна, користувач має можливість отримати доступ до інших об'єктів і командам меню, поки форма відкрита. Якщо спливаюча форма є немодального, не можна отримати доступ до будь-яких інших об'єктів або командам меню, поки форма відкрита. Користувач повинен виконати будь-яку дію, щоб фокус був переключений на іншу форму (або вікно).

5. За *формою подання інформації* екранні форми можуть містити символьну інформацію, ділову графіку, інформацію, представлену в мультимедійній формі. По виконуваних функцій розрізняють форми введення, виведення, керуючі, змішані. Призначення кожного виду цих форм ясно з їх назви.

Програми з повною реалізацією віконного інтерфейсу окремо працюють з окремими підзадачами в різних вікнах. Така програма може одночасно відкривати / працювати з декількома документами, розміщуючи їх в окремі субвікна (наприклад, багатовіконний редактор з документом в кожному вікні). Організацію цих субвікон в подібних програмах реалізують декількома способами:

- одновіконний режим (SDI)
- багатовіконний режим (MDI, TDI)
- псевдобагатовіконний режим (PMDI)

Також одновіконний режим може підтримувати систему фреймів, при якій загальне вікно розбито на кілька функціонально незалежних областей, фреймів (кватирок)(рис. 4.3.).

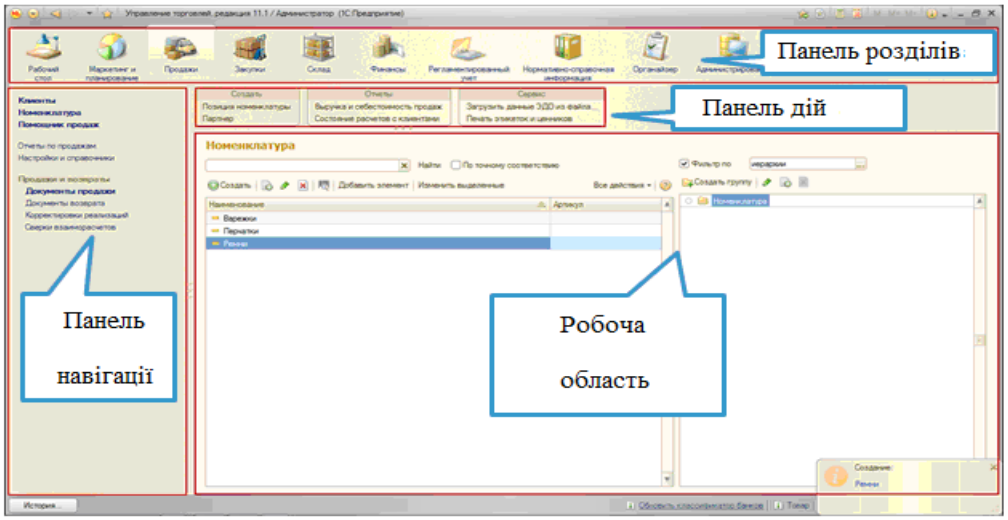
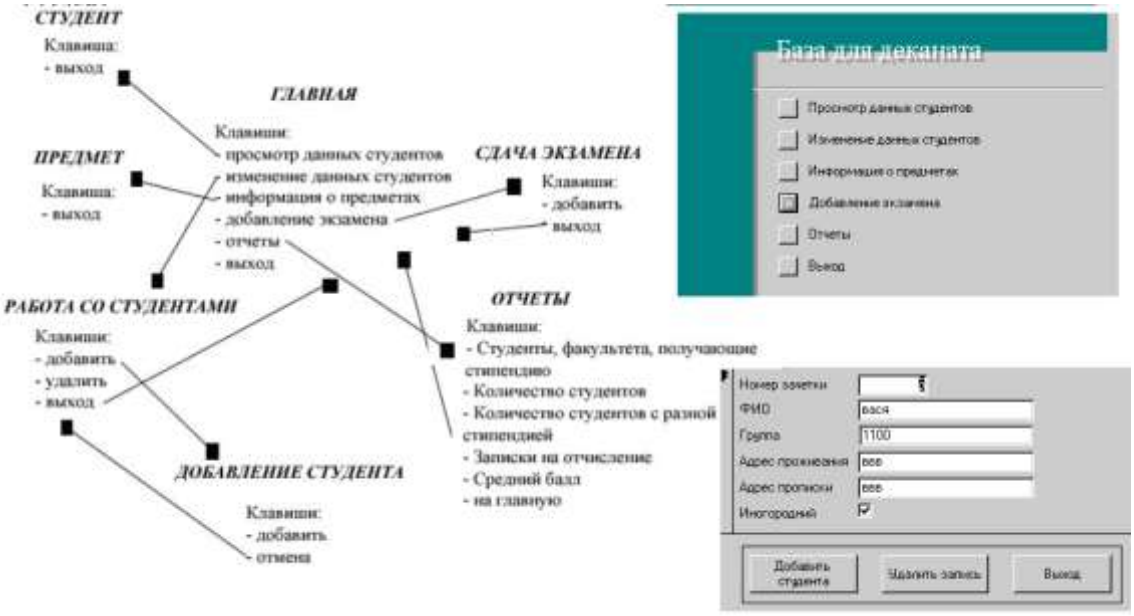


Рис. 4.3. Приклад розбиття екрану на зони

Приклад структури багатовікового програмного додатку наведений на рис. 4.4.



а) б)

Рис. 4.4. Приклад структури ієрархії екранних форм та їх представлення для додатку обліку студентів
а) структура ієрархії;
б) головна та підпорядкована екранні форми

Діалогове вікно – в графічному інтерфейсі є спеціальним елементом інтерфейсу, призначеним для виведення інформації та (або) отримання відповіді від користувача. Здійснює двосторонній «діалог» між користувачем і ПК.

Серед діалогових вікон можна виділити:

- головне – вікно, яке здійснює управління програмним продуктом і з’являється безпосередньо при запуску програми;
- підпорядковане — вікно, яке викликається з головного і призначене для управління певною групою дій.

Діалогові вікна бувають модальними і немодальними, в залежності від блокування або можливості взаємодії користувача з додатком (або системою в цілому) до отримання відповіді від нього. Модальне вікно (modal) блокує роботу батьківського вікна. Батьківське вікно повертає активність лише після закриття модального вікна.

Вікно додатку. Додатками прийнято називати прикладні програми. Кожна програма має головне вікно. В ході роботи з додатком можуть відкриватися додаткові підлеглі вікна.

4.3.2. Створення головного вікна засобами Visual Studio 2017 (C++)

Після установки системи головну екранну форму можна створити виконуючи наступні кроки:

1. Вибираємо пункт меню: File > New > Project... (^+**N**). У вікні, що зявиться, в полі Installed > Visual C++ > CLR > CLR Empty Project вводимо назву проекту.
2. Додаємо головне вікно. Є два рівносильних шляху досягнення цієї мети. Перший: у "Solution Explorer" правою кнопкою миші на назві проекту, у спливаючому контексному меню Add> New Item ... Другий спосіб: в головному меню вибираємо Project> Add New Item ... Або просто натискаємо ^ + A (рис. 4.5.).

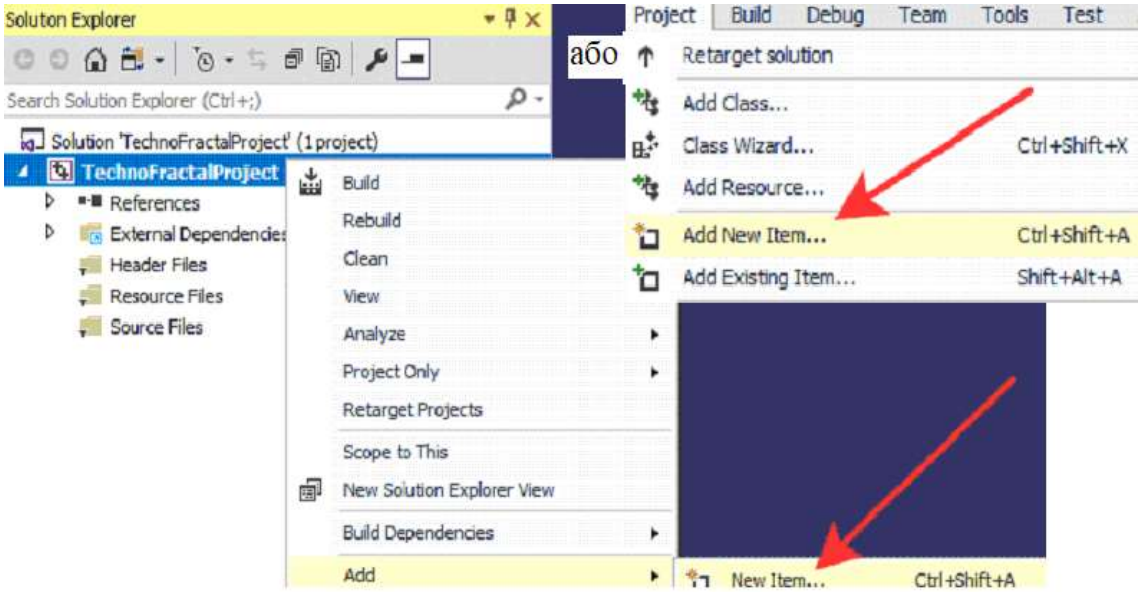


Рис. 4.5. Вибір нового компонента

У вікні Visual C++ > UI > Windows Form вибираємо компонент «форма»: (рис. 4.6.).

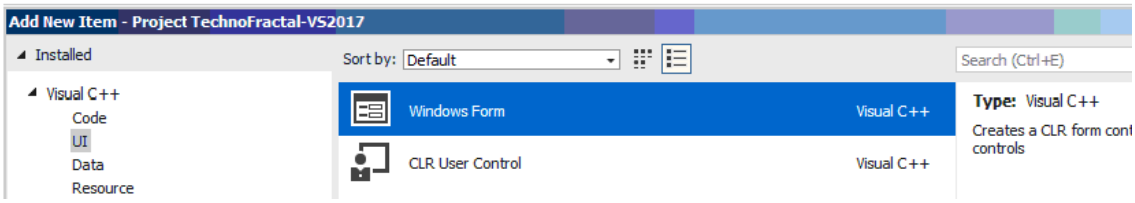


Рис. 4.6. Додавання вікна

Головна форма програми створена. На деяких комп'ютерах в даний момент можливо вискакуванню помилки 0x8000000A, в цьому випадку потрібно просто закрити вкладку.

3. Внести форму в створювану програму. Для цього в "Solution Explorer" правою кнопкою миші на назві проекту, у спливаючому контексному меню вибрати Properties. У вікні, що відкриється, виконати дві дії.

• Linker > System > SubSystem, із списку, що розкриється, вибрати "Windows (/SUBSYSTEM:WINDOWS)" (рис. 4.7).

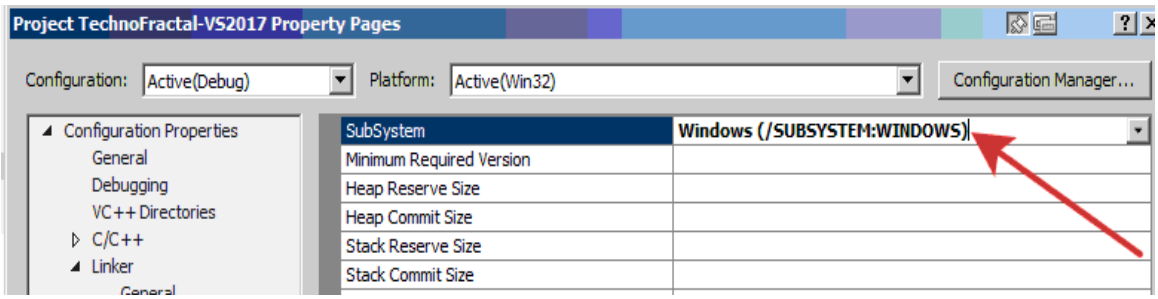


Рис. 4.7. Вибір підсистеми «Вікна»

• Linker > Advanced > Entry Point. В пусте поле вписати "main" *(без лапок)* (рис. 4.8.).

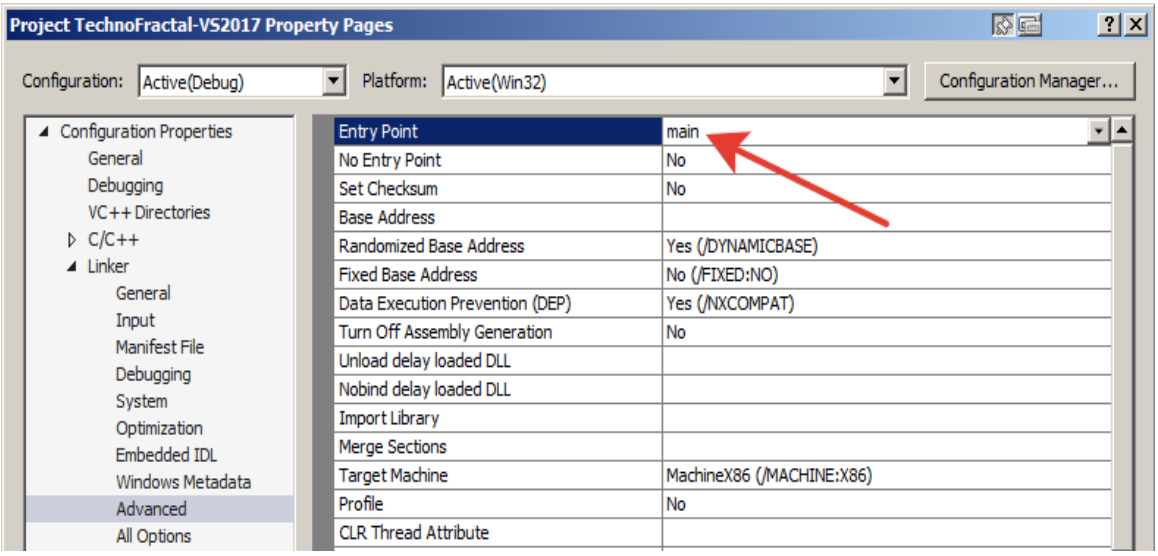


Рис. 4.8. Оголошення вікна як головного

4. В "Solution Explorer" подвійним клацанням відкрити в редакторі файл MyForm.cpp. Скопіювати в нього текст

```
#include "MyForm.h"
using namespace System;
using namespace System::Windows::Forms;
 [STAThread]
void main(array<String^>^ args) {
    Application::EnableVisualStyles();
    Application::SetCompatibleTextRenderingDefault(false);
```

```
Project1::MyForm form;
Application::Run(%form);}
```

Замінивши Project1 на назву вашого проекту (рис. 4.9).

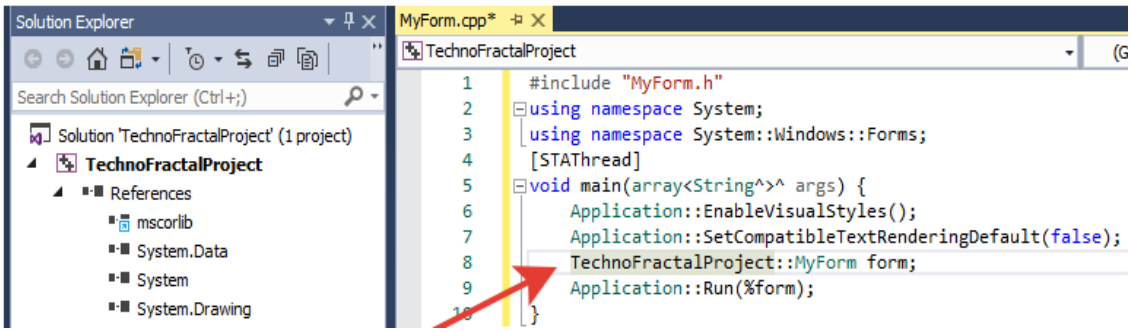


Рис. 4.9. Оголошення нової форми

Тепер проект компілюється і запускається. Але якщо у вас раніше вискакувала 0x8000000A, то швидше за все вам доведеться перезапустити Visual Studio і перезавантажити в ньому проект. Далі помилка ні в чому не проявиться.

Для зміни властивостей щойно створеного елемента інтерфейсу клацніть на ньому правою кнопкою і в контекстному меню виберіть, відповідно, Properties. Для того, щоб додати на нашу щойно створену форму нові елементи, знадобиться панель Toolbox. Корисно запам'ятати гарячу клавішу ^! X.

4.3.3. Розробка багатовіконного інтерфейсу (MDI)

Додатки з багатодокументним інтерфейсом (MDI) дозволяють відображати кілька документів одночасно, при цьому кожен документ відображається в окремому вікні (рис. 4.10). Додатки MDI часто мають пункт меню «вікно» з підменю для перемикання між вікнами або документами.

Розглянемо деякі аспекти розробки багато віконного інтерфейсу.

- Створення батьківської MDI-форми під час розробки. Для цього необхідно виконати наступні кроки:

1. Створити проект програми Windows в Visual Studio.
2. У вікні властивостей IsMdiContainer встановити значення true. При цьому форма призначається в якості MDI-контейнера для дочірніх вікон.
3. Перетягніть елемент управління MenuStrip з панелі елементів в форму. Створіть пункт меню верхнього рівня – для властивості Text задайте значення & File, пункти меню повинні називатися & New і & Close. Також створіть пункт меню верхнього рівня & Window. Перше меню буде створювати і приховувати пункти меню під час виконання, а друге – буде відслідковувати відкриті

дочірні MDI-вікна. На цьому етапі ви створили батьківське MDI-вікно.

4. Натисніть клавішу F5 для запуску програми.

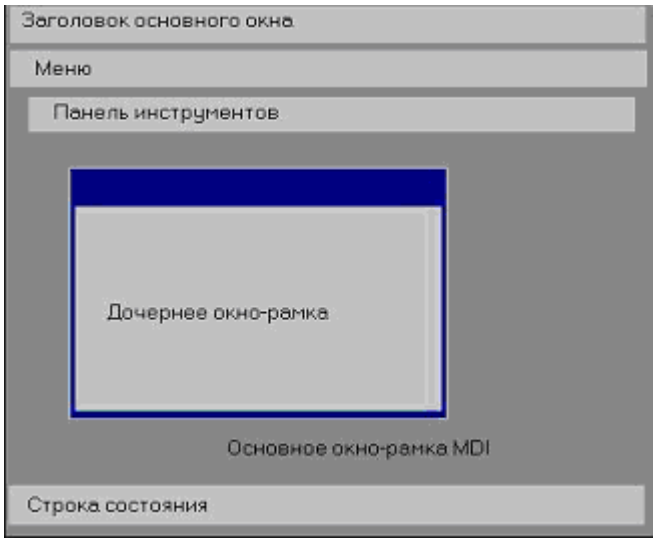


Рис. 4.10. Взаємозв’язок вікна-рамки та вікна відображення в MDI додатку.

- Створення дочірньої MDI-форми

1. Створити новий проект Windows Forms. У вікні Властивості форми задайте для властивості IsMdiContainer значення true, а для властивості WindowsState значення Maximized. При цьому форма призначається в якості MDI-контейнера для дочірніх вікон.
2. З Toolbox перетягніть елемент керування MenuStrip в форму. Дайте властивості Text значення File.
3. Натисніть кнопку з трьома крапками (...) поряд з потрібними Items і натисніть кнопку Додати, щоб додати два дочірніх пункту меню. Дайте властивості Text цих елементів значення New і Window.
4. У браузері рішень клацніть проект правою кнопкою миші і виберіть пункти Додати> Новий елемент.
5. У діалоговому вікні Додавання нового елемента виберіть Windows Form (в Visual Basic або в Visual C #) або Windows Forms додаток (.NET) (в Visual C ++) в області шаблони. В поле ім'я введіть ім'я форми Form2. Натисніть кнопку Відкрити, щоб додати форму в проект. Ця форма буде шаблоном для дочірніх форм MDI.

   Відкриється конструктор Windows Forms, що відображає Form2.

6. Перетягніть елемент управління RichTextBox з панелі елементовв форму.

7. У вікні Властивості задайте для властивості Anchor значення зверху, зліва, а для властивості Dock – значення Fill.

В результаті елемент управління RichTextBox буде цілком заповнювати область дочірньої форми MDI, навіть якщо її розміри зміняться.

8. Двічі клацніть Новий елемент меню, щоб створити для нього обробник подій Click.

9. Вставте код, аналогічний наведеному нижче, щоб створити нову дочірню форму MDI, коли користувач клацне Новий елемент меню.

```
C#      protected void MDIChildNew_Click(object sender,
System.EventArgs e){
            Form2 newMDIChild = new Form2();
            // Set the Parent Form of the Child window.
            newMDIChild.MdiParent = this;
            // Display the new form.
            newMDIChild.Show();}
```

В C++додайте директиву `#include` у верхній частині Form1. h:

```
#include "Form2.h"
```

9. У списку у верхній частині вікна "Властивості" виберіть смугу меню, відповідну смузі меню "файл", і задайте для властивості MdiWindowListItem значення у вікні ToolStripMenuItem. Це дозволяє меню "вікно" підтримувати список відкритих дочірніх вікон MDI з галочкою поруч з активним дочірнім вікном.

10. Натисніть клавішу F5 для запуску програми. Вибравши пункт "створити" в меню Файл, можна створити дочірні MDI-форми.

4.4. Розробка таблиць та стандартних бланків (шаблонів)

Часто в програмних додатках необхідно реалізувати процедури, пов’язані з вводом / виведенням інформації, яка представляється шаблонами або паттернами у вигляді таблиць та стандартними бланками. З метою ефективного та ергономічного використання додатків в реалізація таких документів повинна відповідати їх прийнятому вигляду та наповненню (рис. 4.11,4.12).

Для конструювання стандартних документів можна застосовувати компоненти бібліотеки графічного інтерфейсу представляючи їх у вигляді окремих вікон або об’єднуючи у фрейми (frame). Для дизайну шаблону необхідно чітко визначити:

- яка інформація повинна бути представлена (поля);
- як вона повинна бути представлена;
- де вона повинна бути розташована на документі.

Також необхідно розробити сценарій управління шаблоном (заповнення полів, відображення та ін.) та розробити бібліотеку його програмної реалізації.

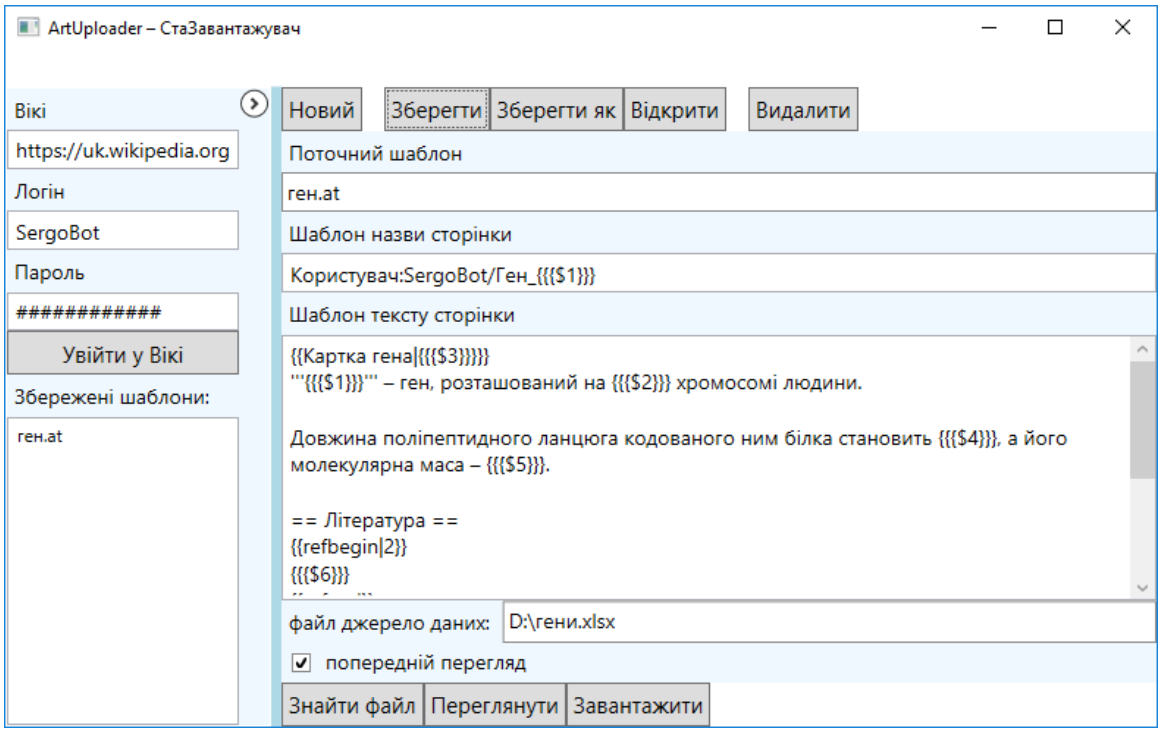


Рис. 4.11.Приклад шаблону

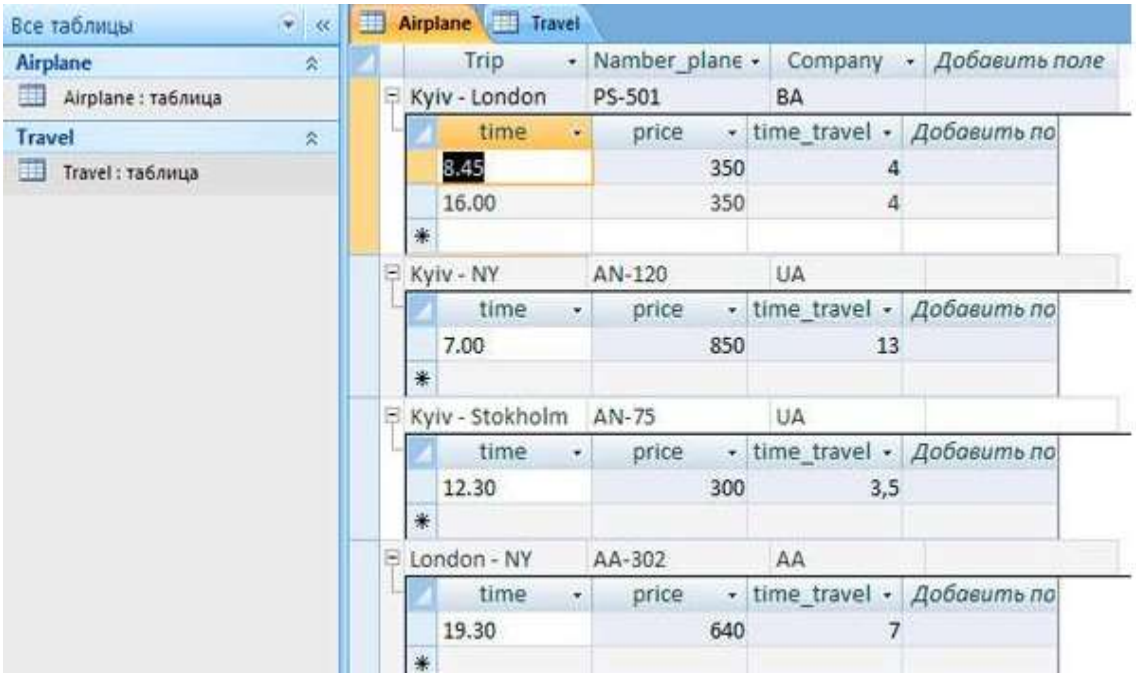


Рис. 4.12. Приклад таблиць бази даних

Контрольні питання

1. Яка параметри даних необхідно визначити для реалізації діалогу?
2. Перерахуйте відомі Вам способи вводу інформації.
3. Які способи та засоби виводі інформації Ви знаєте?
4. Який тип обчислювального процесу доцільно взяти за основу реалізації процедури вводу даних?
5. Що таке синтаксис та семантика інтерфейсу користувача?
6. Які основні компоненти виділяють для реалізації користувацького інтерфейсу?
7. Що таке вікно або екранна форма?
8. Які існують режими роботи з екранними формами?
9. Що таке діалогове вікно?
10. Чим відрізняється модальне вікно від інших?
11. Яке діалогове вікно називають головним?
12. Як називається частина екранного вікна, яка складається з певних елементів?
13. Які компоненти системи Visual Studio необхідні для створення головного графічного вікна інтерфейсу?
14. З яких кроків складається процес створення головного вікна додатку?
15. Що таке MDI?
16. В чому полягає різниця між батьківським та дочірнім вікном?
17. Як створити батьківське MDI-вікно?
18. Як створити дочірнє MDI-вікно?
19. Що називають шаблоном інтерфейсу користувача?
20. Які Ви знаєте способи реалізації шалонів?
21. Що включає процес розробки та реалізації шаблонів?

Лекція 5 Проектування і реалізація компонентів інтерфейсу

5.1. Діалог типу «питання-відповідь»

Структура діалогу типу «питання-відповідь» (Query and Answer - Q & A) заснована на аналогії зі звичайним інтерв'ю. Система бере на себе роль інтерв'юера і отримує інформацію від користувача у вигляді відповідей на питання. Спочатку саме цей режим називали діалоговим.

Діалог типу Q & A має один суттєвий недолік. Навіть якщо введення відбувається досить швидко, для користувача, який вже знає, які питання задає система і які відповіді потрібно на них давати, відповідати на всю серію питань досить утомливо. Проте, така структура особливо доречна при реалізації діалогу з множиною «відгалужень», коли на кожне питання передбачається велике число відповідей, кожен з яких впливає на те, яке питання буде поставлено наступним.

Діалог повинен підкорятися певним правилам:

- учасники діалогу повинні розуміти мову один одного;
- порядок висловлювань в діалозі строго визначений: "питання - відповідь";
- чергове висловлювання повинно враховувати як загальний контекст діалогу, так і останню інформацію, отриману від співрозмовника.

Тип діалогу визначає, хто з "співрозмовників" управляє процесом обміну інформацією.

Розрізняють два типи діалогу:

- керовані програмою;
- керовані користувачем.

Діалог, керований програмою, передбачає наявність жорсткого, лінійного або деревовидного, тобто що включає можливі альтернативні варіанти, сценарію діалогу, закладеного в програмне забезпечення. Такий діалог зазвичай супроводжують великою кількістю підказок, які уточнюють, яку інформацію необхідно вводити на кожному кроці.

Діалог, керований користувачем, має на увазі, що сценарій діалогу залежить від користувача, який застосовує систему для виконання необхідних йому операцій. При цьому система забезпечує можливість реалізації різних призначених для користувача сценаріїв.

Схема діалогової системи може бути представлена наступним чином (рис. 5.1) :

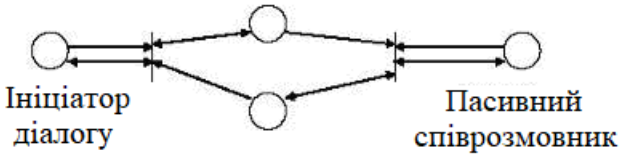


Рис. 5.1. Схема діалогової системи.

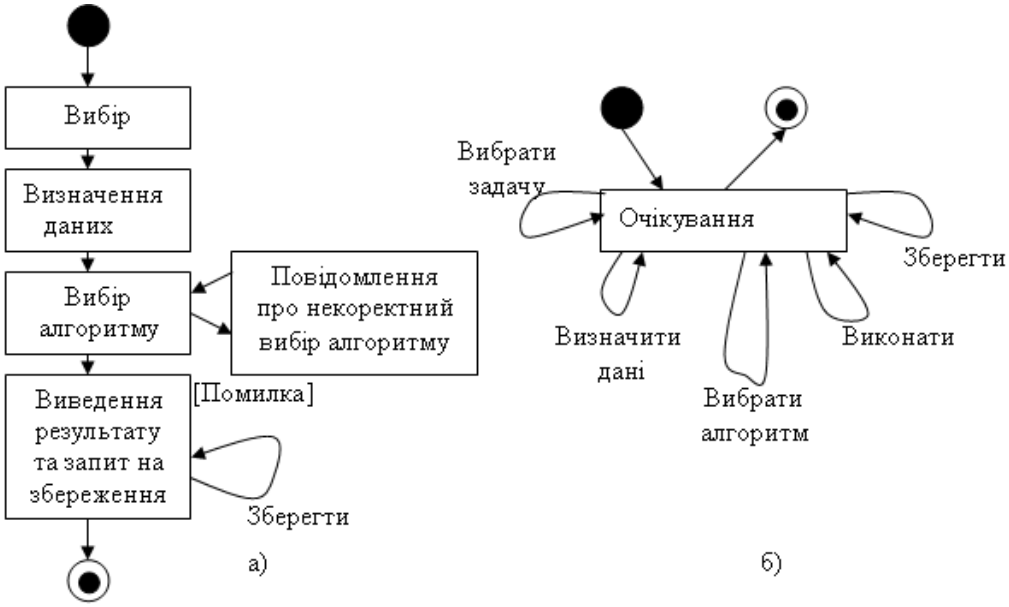


Рис. 5.2. Графи абстрактного діалогу:
а) – діалог, управляєємий системою;
б) – діалог, управляємий користувачем

Вимоги до діалогу типу «питання-відповіь»

Добре організований діалог типу питання відповідь повинен задовольняти наступним вимогам:

1. Природність – це спосіб взаємодії, який не вимагає від користувача змінювати свої традиційні способи вирішення задачі. Стиль ведення діалогу повинен проводитись у вигляді розмови, фрази не вимагати додаткових пояснень, порядок

запитів повинен відповідати встановленій послідовності вирішення задачі.

2. Послідовність – це так логічна взаємодія, яка гарантує, що якщо користувач освоїв одну частину додатку, він не заплутається у використанні іншої. Всі фрази повинні мати певний порядок і використовувати однакові фрази, терміни та позначення. Стандартні відповіді повинні відповідати прийнятим стандартам, наприклад, клавіша F1 для всієї програми повинна відповідати запиту на додаткову інформацію. Для відображення дати повинен бути визначений один формат.
3. Стислість – це принцип, який вимагає стислого формулювання запитів (не більше 40 символів), а користувач повинен вводити мінімум інформації. Не треба вимагати інформацію, яка може бути сформована автоматично.
4. Підтримка користувача – це міра допомоги користувачу при його роботі з додатком. Основними аспектами підтримки є:
   - кількість та якість інструкцій;
   - характер повідомлень про помилки;
   - підтвердження дій.
5. Гнучкість – це адаптація діалогу до користувачів різного рівня підготовки.

Реалізація діалогу типу «питання-відповідь»

Реалізувати діалог типу «питання-відповідь» в середовищі Visual Studio C можна за допомогою наступних базових компонентів:

- Label – компонент призначений для відображення текстової інформації. Задати текст, який відображається в полі компонента, можна як під час розробки форми, так і під час роботи програми, присвоївши значення властивості Text. Щоб в поле компонента Label вивести числове значення, це значення треба перетворити в рядок. Зробити це можна за допомогою методу ToString.
- TextBox – компонент призначений для введення даних з клавіатури. Залежно від настройки компонента, в поле редагування можна ввести одну чи кілька рядків тексту.

- NumericUpDown – компонент призначений для введення числових даних. Дані можна ввести в поле редагування шляхом набору на клавіатурі або змінити вже введені дані за допомогою командних кнопок Збільшити і Зменшити, які знаходяться праворуч від поля редагування.
- Button – компонент являє собою командну кнопку. Може застосовуватись для ініціалізації подій.

Приклади використання компонентів наведені на рис. 5.3.

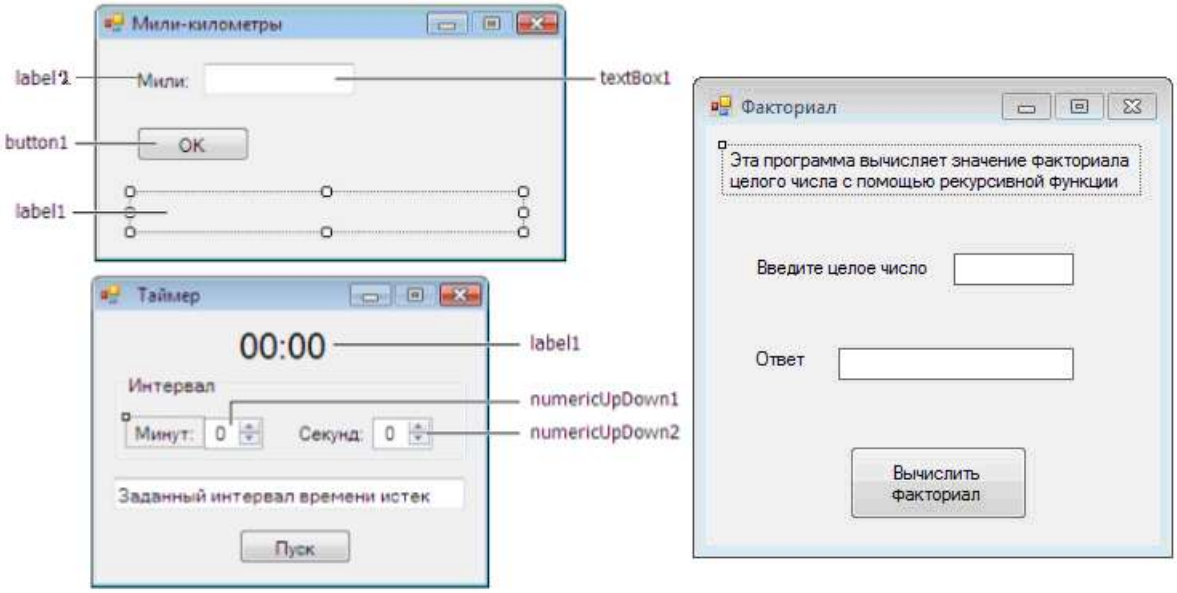


Рис. 5.3. Приклади використання компонентів Label, TextBox, NumericUpDown і Button

5.2. Діалог на основі меню

Меню є найбільш популярним варіантом організації запитів на введення даних під час діалогу, керованого комп'ютером.

Меню можна класифікувати за різними ознаками:

1) про виконанню:

- текстове;
- графічне;

2) по функціям:

- головне меню додатка;
- спливаюче меню;
- контекстне меню;
- системне меню;

3) по розташування:

- стаціонарне;
- спливаюче.

Існує кілька основних форматів уявлення меню на екрані:

- список об'єктів, які обирають прямою вказівкою, або зазначенням номера (або мнемонічного коду);
- меню у вигляді блоку даних;
- меню у вигляді рядка даних;
- меню у вигляді піктограм.

Користувач діалогового меню може вибрати потрібний пункт, вводячи текстовий рядок, який ідентифікує цей пункт, вказуючи на нього безпосередньо або переглядаючи список і вибираючи з нього. Система може виводити пункти меню послідовно при цьому користувач вибирає потрібний йому пункт натисканням клавіші.

Меню у вигляді рядка даних може з'являтися вгорі або внизу екрану і часто залишається в цій позиції протягом усього діалогу. Таким чином, за допомогою меню зручно відображати можливі варіанти даних для введення, доступних в будь-який час роботи з системою.

Якщо в системі є досить велика різноманітність варіантів дій, організовується ієрархічна структура з відповідних меню (рис. 5.4).

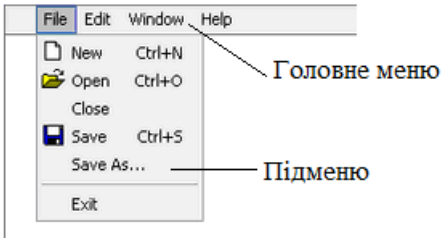


Рис. 5.4. Приклад ієрархічної структури меню

Додаткові меню у вигляді блоків даних «спливають» на екрані в позиції, яка визначається поточним становищем покажчика, або «випадають» безпосередньо з рядка меню верхнього рівня. Ці меню зникають після вибору варіанта.

Меню у вигляді піктограм представляє множину об'єктів вибору, розкиданих по всьому екрану; часто об'єкти вибору містять графічне представлення варіантів роботи.

Меню можна з рівним успіхом застосовувати для введення як керуючих повідомлень, так і даних. Прийнятна структура меню залежить від його розміру та організації, від способу вибору пунктів меню і реальної потреби користувача в підтримці з боку меню.

Структура типу меню є найбільш природним механізмом для роботи з пристроями вказівки і вибору: меню являє собою зображення тих об'єктів, які вибираються користувачем. Якщо діалог складається виключно з меню, можна реалізувати послідовний інтерфейс, при якому користувач застосовує тільки пристрої для вказівки; проте така сталість рідко можна досягти на практиці. Слід зазначити, що хоча робота з цими пристроями не вимагає професійного володіння клавіатурою, для підготовленого користувача це не найшвидший спосіб вибору з меню. Замість вказівки користувач може повідомити про свій вибір введенням відповідного ідентифікатора.

Меню - це найбільш зручна структура діалогу для непідготовлених користувачів; жорстка черговість відкриття і ієрархічна вкладеність меню може викликати роздратування професіонала, уповільнювати його роботу. Традиційна структура меню недостатньо гнучка і не в повній мірі узгоджується з методами адаптації діалогу, такими, наприклад, як випереджаюче введення, за допомогою якого можна прискорити темп роботи підготовленого користувача.

Реалізація діалогу на основі меню

Реалізувати меню в середовищі Visual Studio C можна за допомогою наступних базових компонентів:

- MenuStrip – компонент являє собою головне меню програми. Після того як в форму буде додано компонент MenuStrip, у верхній частині форми з'являється рядок меню, на початку якої знаходиться область введення. Щоб створити перший пункт меню, треба зробити клацання в області введення тексту і ввести назву пункту меню, наприклад, *Файл*. Як тільки буде введена перша буква, праворуч і знизу від області введення з'являться ще дві області введення тексту. У нижню область треба ввести назву команди меню, наприклад, *Новий*, а в праву - назва наступного пункту меню, наприклад, *Параметри* (рис. 5.5).

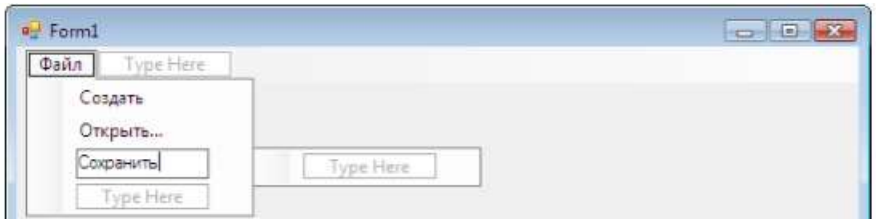


Рис. 5.5. Створення головного меню

При розробці головних меню, керуються наступними правилами:

- кількість елементів верхнього рівня повинна не перевищувати 25;
- меню ніколи не повинно перевищувати 600 пікселів у висоту.
- спливаюче меню є допустимими.
- спливаюче меню повинно містити принаймні три елементи і не більше семи.
- спливаюче меню повинно мати тільки один рівень глибини
- деякі пункти меню Visual Studio є каскадними підменю, але цей шаблон не рекомендується.

- CheckBox – компонент прапорець, який може знаходитися в одному з двох станів: обраному або невибраний. Поруч з прапорцем, як правило, знаходиться пояснювальний текст. Компоненти CheckBox зазвичай використовують для вибору кількох опцій з ряду можливих.
- RadioButton – компонент являє собою перемикач, стан якого залежить від стану інших перемикачів (компонентів RadioButton). Зазвичай компоненти RadioButton об'єднують в групу (досягається це шляхом розміщення кількох компонентів в поле компонента GroupBox). На відміну від CheckBox у кожен момент часу тільки один з перемикачів групи RadioButton може перебувати в обраному стані (можлива ситуація, коли ні один з перемикачів не вибрано). Стан компонентів, що належать одній групі, не залежить від стану компонентів, що належать іншій групі.
- ComboBox – компонент являє собою комбінацію поля редагування і списку, що дозволяє вводити дані шляхом набору на клавіатурі або вибору значення в списку. Список, який

відображається в полі компонента, можна сформувати під час створення форми або під час роботи програми.

- ListBox – компонент являє собою список, в якому можна вибрати потрібний елемент. Список можна сформувати під час створення форми або під час роботи програми.

5.3. Діалог на основі команд

Для командного інтерфейсу характерно взаємодія користувача з ЕОМ за допомогою командного рядка, в якому вводяться команди певного формату, а потім передаються до виконання. Командний інтерфейс реалізований у вигляді пакетної технології та технології командного рядка.

Робота з командним інтерфейсом полягала в наступному:

- користувач вводив команду за допомогою послідовності символів (командного рядка);
- комп'ютер зіставляв команду, що надійшла, з наявним в його пам'яті набором команд;
- виконується дія, яка відповідає заданій команді.

До переваг інтерфейсу командного рядка відносять:

- низькі вимоги до апаратних засобів – мінімальним набором для роботи є клавіатура і символьне пристрій виведення або термінал;
- висока ступінь уніфікації – взаємодія забезпечується через введення і виведення символів, який часто реалізується через файловий ввід-висновок;
- широка можливість інтеграції програм – за допомогою використання командного інтерпретатора і перенаправлення вводу-виводу.

Недоліками командного інтерфейсу вважають:

- погану наочність інтерфейсу – необхідно пам'ятати команди або використовувати довідник;
- обмежені можливості виведення інформації – відсутня графіка.

Багато командних мов підтримує макроси, які розширюють функціональні можливості діалогу, скорочуючи при цьому кількість команд, що вводяться. Макрос містить кілька командних рядків. При

зверненні до нього в процесі діалогу окремі рядки команд макросу виконуються оди за одним, як якщо б вони вводилися з клавіатури.

5.4. Представлення структури діалогу

Для кращого розуміння реалізації та роботи інтерфейсу,як для програміста так і користувача, створюється схема навігації у вигляді графу або граф, який називається мережею переходів.

Призначення схеми навігації – висловити основні шляхи призначеного для користувача інтерфейсу в системі. Це основні шляхи через екрани системи, але не обов'язково всі можливі шляхи. Її можна представити як карту доріг призначеного для користувача інтерфейсу системи.

Схема навігації служить в якості фону і зв'язку між окремими розкадровки. Розкадровки описують навігацію переміщень користувача за елементами призначеного для користувача інтерфейсу для виконання системних функцій, а схема навігації визначає допустимі шляху навігації. Схема навігації висловлює структуру користувальницького інтерфейсу системи, а розкадровка – динаміку.

Схема навігації полегшує оцінку того, скільки "клацань" повинен зробити користувач, щоб досягти певного екрану або певної дії.

Для схеми навігації можна використовувати різні уявлення. Нижче наведені деякі приклади:

- ієрархічна діаграма в вигляді "дерева", кожен рівень якої показує число клацань для досягнення певного елемента користувальницького інтерфейсу;
- зображення у вільній формі з одними значками.

Обрані представлення і всі рішення по налаштуванню слід задокументувати, вони відносяться до проекту рекомендацій.

При розробці схематичного представлення структури інтерфейсу необхідно дотримуватись певних правил при виборі позначень:

- зміст повинен відображати модель представлень з точки зору користувача;

- необхідно використовувати мову аудиторії для якої призначений додаток;
- необхідні не просто позначення, а системні послідовні позначення.

творення схеми починається з розробки загальної структури системи («вид з висоти пташиного польоту»), тобто необхідно виділити окремі функціональні блоки і визначити, як саме ці блоки зв'язуються між собою. Під окремим функціональним блоком будемо розуміти функцію / групу функцій, пов'язаних з призначенням або області застосування в разі програми і групу функцій / фрагментів інформаційного наповнення.

Проектування загальної структури складається з двох паралельно відбуваються: виділення незалежних блоків і визначення зв'язку між ними.

Виділення незалежних блоків. Для цієї роботи важко дати які-небудь конкретні рекомендації, оскільки дуже багато що залежить від проектованої системи. Проте, можна с упевненістю рекомендувати уникати приміщення в один блок більше трьох функцій, оскільки кожен блок в системі буде укладено в окремий екран або групу керуючих елементів. Перевантажувати інтерфейс небезпечно. Результатом цієї роботи має бути список блоків з необхідними поясненнями.

Визначення смислового зв'язку між блоками. Існує три основних види зв'язку між блоками. Це логічний зв'язок, зв'язок за поданням користувачів і процесуальний зв'язок. Логічний зв'язок визначає взаємодію між фрагментами системи з точки зору розробника (суперкористувача). Користувачі мають свою думку про систему, і ця думка теж є важливим видом зв'язку. Нарешті, процесуальний зв'язок описує нехай не цілком логічну, але природню для наявного процесу взаємодію: наприклад, логіка безпосередньо не командує людям спочатку приготувати обід, а потім з'їсти його, але зазвичай виходить саме так. Всі три типи взаємозв'язку повинні бути заздалегідь передбачені при конструюванні системи.

Результатом проектування є схема, побудована, наприклад, в MS Visio або іншими інструментальними засобами (рис. 5.5).

Рис 5.6. Приклад навігації додатку.

На наведеному рис. 5.6 кожний прямокутник представляє екранну форму.

Повна документація може представляти собою ієрархічно зв’язаний альбом документів, де кожний рівень ієрархії є деталізацією компонентів схеми вищого рівня.

5.5. Створення простого віконного додатку з графічним інтерфейсом у Visual Studio C++

Для створення простого додатку виконаємо наступні кроки:

1. Спочатку створимо проект. Відкриємо Visual Studio, далі натиснемо Файл → Створити → Проект, далі вибираємо пункт CLR і відзначаємо Додаток Windows Forms, даємо ім'я проекту, наприклад factr і тиснемо Ok.
2. Після того як ми створили проект, повинна з'явитися порожня форма.

Праворуч повинна з'явитися панель елементів, якщо її немає, то можна включити її в меню Вид → Панель Елементів або натисканням гарячих клавіш - Ctrl + Alt + X (рис. 5.7).

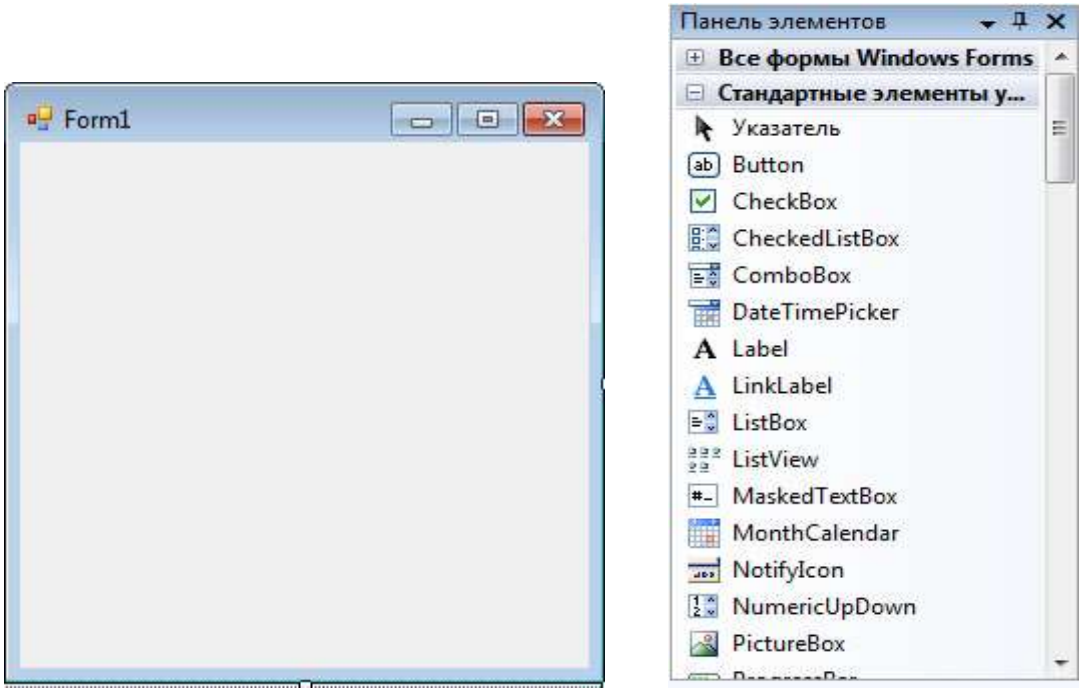


Рис. 5.7. Пуста форма та панель інструментів

3. Перетягніть на форму три елементи Label, Два елементи TextBox, і одну кнопку (Button), розставте елементи приблизно як на рис. 5.8.

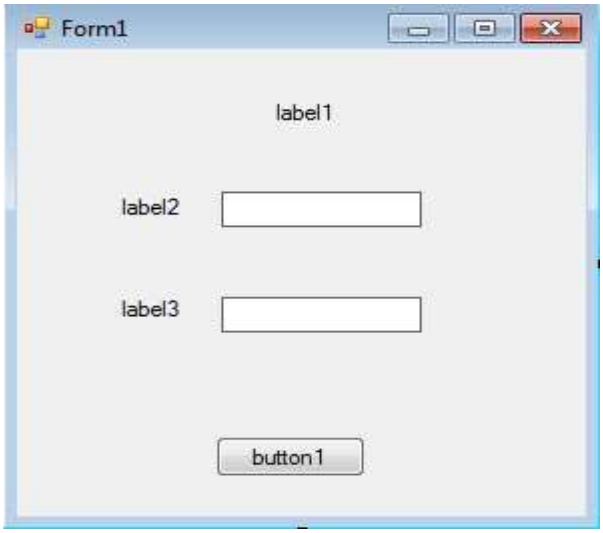


Рис. 5.8 Вигляд форми після встановлення компонентів

4. Тепер потрібно змінити текст написів для елементів Label і Button і Задати відповідні імена класів для елементів TextBox. Для цього виділіть елемент Label1, перейдіть в Панель властивостей, (зазвичай вона знаходиться під панеллю елементів, якщо панель

вимкнена, включіть її в меню Вид -> Диспетчер властивостей) і задайте для даного елемента значення атрибута Text як показано на рис.5.9.

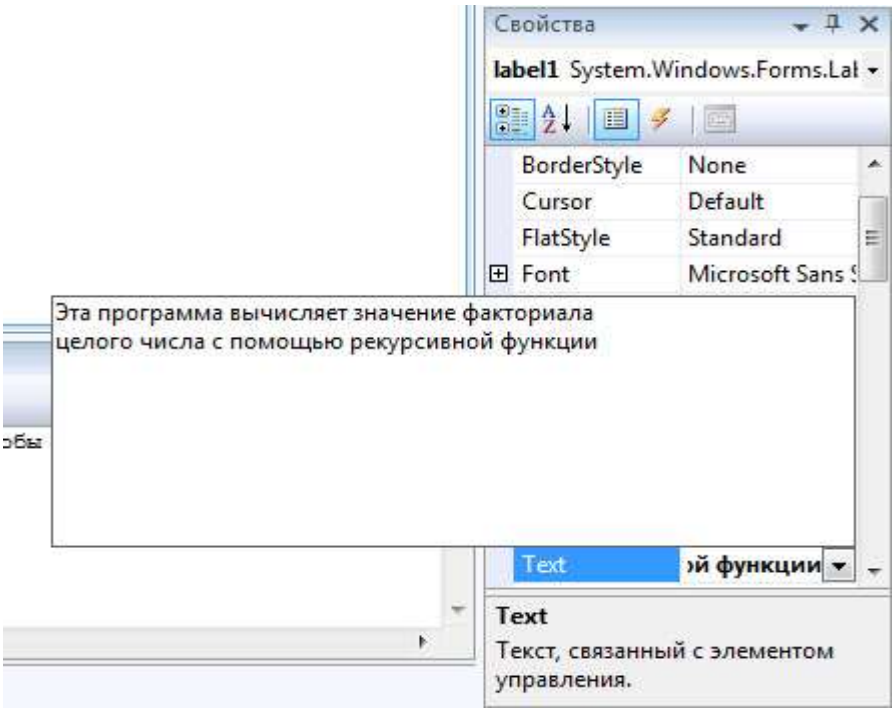


Рис. 5.9 Задання напису на компоненті Label1

Задайте аналогічним чином атрибут Text для всіх елементів Label і кнопки Button. Наша форма прийме вигляд, як на рис. 5.10.

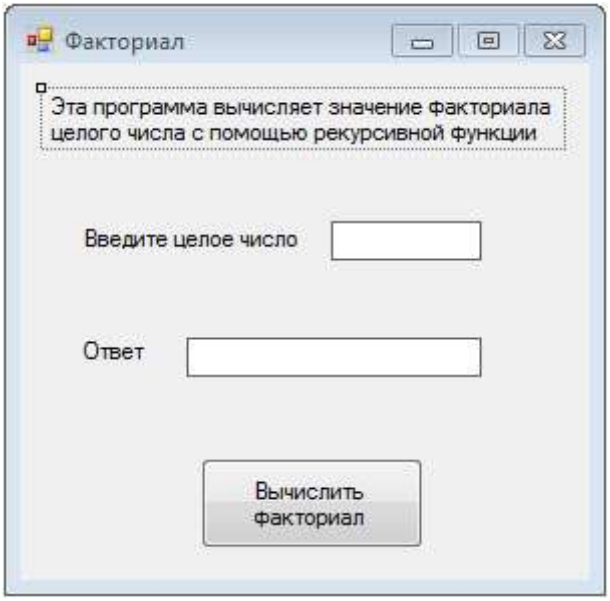


Рис. 5.10. Вигляд налаштованої форми

Тепер задайте для текстових полів (TextBox) Атрибути Name в Панелі Властивостей - для першого поля це буде *num1*, а для другого,

відповідно *num2*. Тим самим ми змінили імена класів для полів TextBox. Отже, каркас форми готовий,

5. Перейдемо до написання обробника подій для кнопки Button, тобто що буде відбуватися при натисканні на цю кнопку, також ми напишемо саму функцію обчислення факторіала, яку ми будемо використовувати в обробнику.

Насамперед потрібно додати заголовки fact.h в проект, для цього клацаємо правою кнопкою миші в Оглядачі рішень на папці Заголовки, далі меню Додати -> Створити Елемент, вписуємо назву файлу – fact.h і натискаємо Додати. В даному файлі буде міститися прототип функції для обчислення факторіала. Додаємо туди наступний вихідний код.

```
long double fact(int N);
```

Далі підключаємо цей файл до проекту за допомогою директиви #include - Відкриваємо файл factr.cpp і додаємо після рядки #include "stdafx.h" наступний код.

```
#include "fact.h"
```

Тепер заготовочний файл підключений до проекту, далі аналогічним чином створюємо файл вихідного коду fact.cpp в проект. У нього додаємо саму функцію обчислення факторіала цілого числа.

```
#pragma once
#include "stdafx.h"
  long double fact(int N) {
  // если пользователь ввел отрицательное число
  if(N < 0) // возвращаем ноль
  return 0; // если пользователь ввел ноль
  if (N == 0) // возвращаем факториал нуля
  return 1; // Во всех остальных случаях
  else // делаем рекурсию return N * fact(N - 1); }
```

Тепер наш проект містить функцію обчислення факторіала, відкомпілюйте його, якщо помилок немає, то можна приступити до написання обробника подій для кнопки Button.

6. Відкриваємо файл Form.h, в ньому ми бачимо візуальне представлення нашої форми. Клацаємо двічі на кнопку Button і переходимо до вихідного коду.

Після фігурної дужки ставимо абзац і приступаємо до кодування. Будь-який набір символів, які ми вводимо з клавіатури в текстове поле програми є рядком, тому нам потрібно витягти значення з текстового поля і привести його до цілочисельного типу. Спочатку нам необхідно описати і форматувати змінну, в якій буде зберігатися значення числа, для якого повинен бути обчислений факторіал. Додамо в тому місце, де поставили абзац наступний код:

```
int number = System::Convert::ToDouble(num1->Text);
```

Цей рядок инициализирует змінну, яка зберігає в собі число для обчислення його факторіала. Далі нам потрібно буде зробити виклик нашій функції fact зі змінною number в якості єдиного аргументу і занести результат роботи функції в нову змінну factor.

```
double factor = fact(number);
```

І потім перетворити отримане значення назад в рядок і присвоїтити його другому текстовому полю.

```
num2->Text = System::Convert::ToString(factor);
```

Наш простий обробник подій буде виглядати ось так:

```
int number = System::Convert::ToDouble(num1->Text);
double factor = fact(number);
num2->Text = System::Convert::ToString(factor);
```

Тепер можна скомпілювати і запустити готову програму (рис. 5.11). Пам'ятайте, що тип double в C ++ може зберігати число не перевищує! 170.

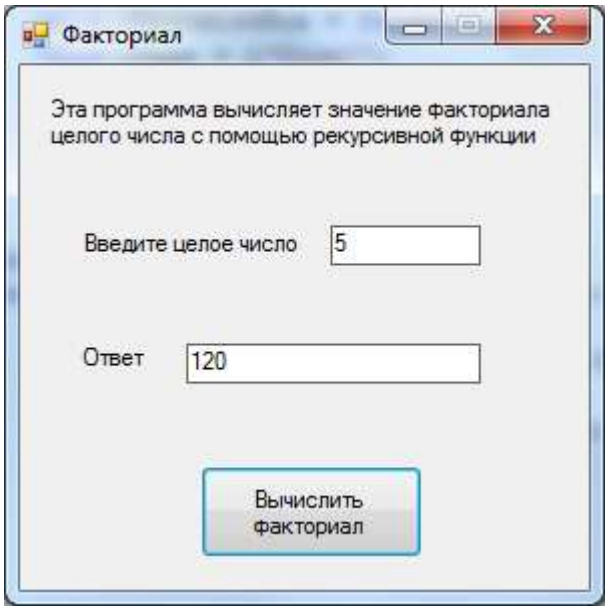
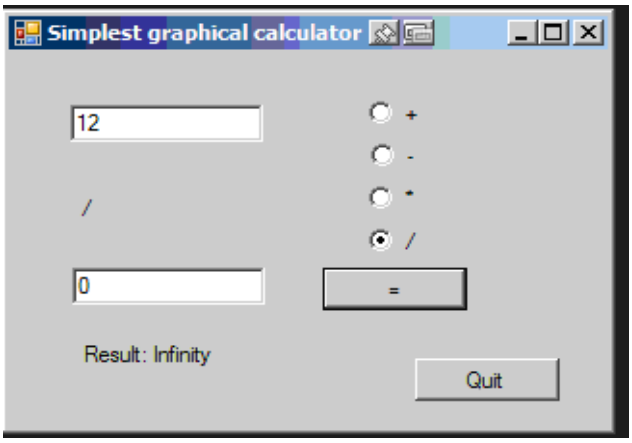


Рис. 5.11 Результат виконання програми

```
private: System::Void button6_Click(System::Object^  sender,
System::EventArgs^  e)   {    Application::Exit();}
private: System::Void radioButton1_CheckedChanged(System::Object^
sender, System::EventArgs^  e)
{    label1->Text = "+";}
private: System::Void radioButton2_CheckedChanged(System::Object^
sender, System::EventArgs^  e)
{    label1->Text = "-";}
private: System::Void radioButton3_CheckedChanged(System::Object^
sender, System::EventArgs^  e)
{    label1->Text = "*";}
private: System::Void radioButton4_CheckedChanged(System::Object^
sender, System::EventArgs^  e)
{    label1->Text = "/";}
private: System::Void
button5_Click(System::Object^
sender, System::EventArgs^  e)
{    char sw =
System::Convert::ToChar(label1-
>Text);
   double a =
System::Convert::ToDouble(textBox1->Text);
   double b = System::Convert::ToDouble(textBox2->Text);
   double r;
   switch (sw) {
   case '+':       r = a + b;
      break;
   case '-':       r = a - b;
      break;
   case '*':       r = a * b;
      break;
   case '/':       r = a / b;
      break;    }
   label2->Text = "Result: " + System::Convert::ToString(r);
}
```



Контрольні питання

1. Що таке діалог Q&A?
2. Якими правилами необхідно керуватись при проектуванні діалогу типу Q&A?
3. Які недоліки має діалог типу Q&A?
4. Які розрізняють види діалогу типу Q&A?
5. Наведіть приклад схематичного представлення діалогу типу Q&A?
6. Які вимоги висуваються до проектування та реалізації діалогу типу Q&A?
7. За допомогою яких візуальних компонентів Visual Studio можна реалізувати діалог типу Q&A?
8. Що таке діалог на основі меню?
9. Які типи меню виділяють?
10. Які існують способи представлення меню на екрані?
11. Які вимоги необхідно враховувати при проектуванні діалогу типу меню?
12. Які існують вимоги до ієрархічних меню?
13. Які переваги має діалог на основі меню?
14. За допомогою яких візуальних компонентів Visual Studio можна реалізувати діалог типу меню?
15. Що таке діалог на основі команд?
16. Які переваги та недоліки має діалог на основі команд?
17. Які схеми застосовують для представлення структури діалогу?
18. Які існують вимоги до позначень на схемах діалогу?
19. Що таке схема навігації?
20. Наведіть приклад графу мережі переходів для діалогу типу:
    а) питання відповідь:
    б) меню.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Скидан С.А. Понятие и сущность эргономики как науки // Придніпровський науковий вісник. Педагогіка середньої та вищої школи. 1998. - №70. - С.1-8.
2. GN1405 Эргономика [Електронний ресурс] https://it.rfei.ru/course/~Ylac/~VfelPrP9
3. Бурлак Г.Н. Безопасность работы на компьютере: Организация труда на предприятиях информационного обслуживания: Учеб. пособие. М.: Финансы и статистика, 1998. 144 с.
4. Моргунов Е.Б. Человеческие факторы в компьютерных системах. М.: Тривола, 1994. 270 с.
5. Калилец Т.В. Эргономика информационных систем: пособие / Т. В. Калилец, В. С. Осипович, И. Ф. Киринович, В. В. Савченко, К. Д. Яшин.– Минск, БГУИР. 2017. 73 с.
6. Антонов А.В. Информация: восприятие и понимание. Киев: Наукова думка. 1988.- 184 с.
7. Ергономіка кінцевого користувача [Електронний ресурс] https://it.rfei.ru/course/~Ylac/~VfelPrP9
8. Ашеров А.Т. Эргономика информационных технологий в примерах и задачах: Учебное пособие / А.Т. Ашеров, Г.И. Сажко, Е.А. Лавров, В.Г. Хоменко, Ю.Н. Полякова. – Горловка: ЧП" Видавництво Ліхтар. 2007. 214с.
9. Коутс Р., Влейминк И. Интерфейс человек-компьютер. М. Мир 1990г. 501с.
10. Минаси М. Графический интерфейс пользователя: секреты проектирования / Пер. с англ. М.: Мир, 1996. 160 с.
11. Проектирование пользовательского интерфейса на персональных компьютерах. Стандарт фирмы IBM / Под ред. М. И. Дадашова. М.: АО «Лев», 1992. - 186 с.
12. Федоров Б.И., Джалишвшш З.О. Логика компьютерного диалога. М.: Онега, 1994. - 240 с.
13. Бондаровская А., Повякель Н. Психолого-эргономическое обеспечение программных средств // Информатика и образование. 1990. - № 5. - С. 68-74.

14. Бурмистров И.В. Гибкий сценарный интерфейс: интеллектуальная пространственная структура диалога // Психологический журнал. 1993. - Т. 14, - № 2. - С. 44-53.
15. Евстигнеев В.Е., Смолян Г.А. Эргономические принципы организации графического представления данных / Эргономика. Тр. ВНИИТЭ. — Вып. 31. Эргономическое обеспечение проектирования и эксплуатации средств автоматизации. -М.: ВНИИТЭ, 1986. С. 47-51.
16. Кроль В.М. Психологические аспекты разработки визуального пользовательского интерфейса нового поколения // Пользовательский интерфейс. -1993. -№3.- С. 17-22.
17. Машбиц Е.И. Психолого-педагогические проблемы компьютеризации обучения. -М.: Педагогика. 1988. -192 с.
18. ISO 18529 Эргономика человеко-компьютерного взаимодействия.
19. ISO 14915 Эргономика программного обеспечения мультимедийных пользовательских интерфейсов.
20. ISO 16071 Эргономика взаимодействия «человек-система».
21. Хаустова Е.Ю. Программирование в Visual C++. Учебное пособие. / Каменский институт им. М.И. Платова, Каменск-Шахтинский. 2016. 40 с.
22. Культин Н. Б. К90 Основы программирования в Microsoft Visual C++ 2010. — СПб.: БХВ-Петербург, 2010. 384 с.
23. Bruce Johnson Professional Visual Studio 2017. . — John Wiley & Sons. 2018. 863 p. [Електронний ресурс] http://ablbook.com/index.php?newsid=21844

Навчальне видання

ГОРДА Олена Володимирівна,

ЕРГОНОМІКА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Конспект лекцій

Редагування та коректура
Компютерне верстання

Підписано до друку Формат
Папір офсетний. Гарнітура Таймс. Друк на різографі.
Ум. друк. арк. Обл.-вид. арк.
Ум.фарбовідб. Тираж прим. Вид. № Зам. №

КНУБА, Повітрофлотський проспект, 31, Київ, Україна, 03680

E-mail: red_instad@ua.fm

Віддруковано в редакційному відділі
Київського національного університету будівництва і архітектури

Свідоцтво про внесення до Державного реєстру суб'єктів видавничої справи
ДК №