Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя
(повне найменування вищого навчального закладу)
Факультет комп'ютерно-інформаційних систем і програмної інженерії
(назва факультету)
Комп'ютерні науки
(повна назва кафедри)

ПОЯСНЮВАЛЬНА ЗАПИСКА

до дипломної роботи

магістр
(освітній рівень)

на тему: **Програмна реалізація репозитарію шаблонів проектування для розробки програмних архітектур**

Виконав: студент 6 курсу, групи СНмз-61
спеціальності 122 «Комп'ютерні науки»
(шифр і назва спеціальності)

		Гупало Т.О.
	(підпис)	(прізвище та ініціали)
Керівник		Боднарчук І.О.
	(підпис)	(прізвище та ініціали)
Нормоконтроль		Мацюк О.В.
	(підпис)	(прізвище та ініціали)
Рецензент		
	(підпис)	(прізвище та ініціали)

м. Тернопіль – 2019

АНОТАЦІЯ

Програмна реалізація репозитарію шаблонів проектування для розробки програмних архітектур // Дипломна робота ОР "Магістр" // Гупало Тарас Олегович // Тернопільський національний технічний університет ім. І. Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра комп'ютерних наук // Тернопіль, 2019 // с. – , рис. – , табл. – , джерел – .

Ключові слова: АЛЬТЕРНАТИВНІ АРХІТЕКТУРИ, ПАТЕРНИ ПРОЕКТУВАННЯ, МОДУЛЬ, ШАР, КАРКАСИ АРХІТЕКТУР, ІНФОРМАЦІЙНА СИСТЕМА.

Метою проектування є розгляд теоретичних та практичних засад створення репозиторію альтернативної архітектури, маючи набір загальних патернів і архітектур що реалізують різні модулі програми.

Метою проєкту є розгляд теоретичних та практичних засад створення та управління репозиторіем патернів і архітектур. Категоризація для побудови рапозиторію для каркасів архітектур і патернів.

Предметом дослідження є програмне забезпечення теоретичні засади оцінки альтернативних архітектур та програмній комплект що формую набір альтернативних архітектур після чого проводиться порівняльна оцінка їх та загальний вибір оптимальнішої архітектури виходячи з експертних оцінок.

ANNOTATION

Quality model development for transport software automatic control system // Diploma paper of Master degree level // Hupalo Taras Olehovych // Ternopil Ivan Puluj National Technical University, Faculty of Computer Information Systems and Software Engineering, Department of Computer Science // Ternopil, 2019 // p. – , Fig. – , Table. – , Refence. – .

Key words: ALTERNATIVE ARCHITECTURES, PATTERN DESIGN, MODULE, BALL, ARCHITECTUR FRAMEWORKS, INFORMATION SYSTEM.

The purpose of the design is to consider the theoretical and practical foundations of creating a repository of alternative architecture, having a set of common patterns and architectures that implement different modules of the program.

The purpose of the project is to review the theoretical and practical foundations of creating and managing a repository of patterns and architectures. Categorization for constructing a repository for frameworks of architectures and patterns.

The subject of the study is the software theoretical basis for the evaluation of alternative architectures and the software package that forms a set of alternative architectures, after which a comparative evaluation of them and the overall choice of the best architecture based on expert estimates.

ЗМІСТ

ВСТУП

Створення сучасних програмних продуктів вимагає від проектувальників враховувати великі об’єми даних, знань, факторів для прийняття ними ефективних рішень. Програмні системи (ПС) є високоінтелектуальним продуктом, що ускладнює формалізацію процесів його проектування, а це, в свою чергу, затрудняє розробку та використання засобів автоматизації цих процесів.

Особливо важливо це на етапах специфікації вимог і проектування архітектури, оскільки ці етапи є першими і виправлення результатів неефективних рішень, прийнятих на них, приводить до великих втрат.

При проектуванні програмних систем широко застосовується компонентна технологія яка базується на вживанні компонентів повторного використання (КПВ), які взяті з раніше виконуваних проектів. Архітектура в цій технології проектується вибором, на основі вимог до ПС, каркасу і заповненням його необхідними компонентами, взятими з репозиторію, або інтернету. Каркас являє собою високорівневу абстракцію проекту ПС і поєднує множину взаємодіючих між собою об'єктів у деяке інтегроване середовище. Розширенням поняття компонента є шаблон (паттерн) – абстракція, що містить у собі опис взаємодії сукупності об'єктів у загальній кооперативній діяльності, для якої визначені ролі учасників і їхня відповідальність. Розроблено велику кількість компонентів, які класифіковані по типам задач і видам застосувань, а також технологія їх використання.

Оскільки в репозиторії патернів, як правило, є декілька компонентів, які реалізують одну і ту ж функцію, то отримаємо певну множину альтернативних архітектур ПС. Для вибору найбільш прийнятного варіанта архітектури необхідно знайти оцінки альтернатив відносно критеріїв якості, при заданих обмеженнях.

На практиці використовується декілька методів оцінювання програмної архітектури.

При створенні даного модуля для подання архітектури була прийнята концепція шарів, розвинута М.Фаулером, в якій функціональність розділена на шари. Корпорація Microsoft розробила технологію, яка базується на даній концепції. В цій технології для кожного шару розроблено набори компонентів (патернів), які реалізують функціональність даного шару. Компоненти згруповані у категорії і призначені для вирішення певних стандартних задач.

Таким чином, кожне програмне застосування, яке буде проектуватись, можна розділити на логічні частини, які відповідають шарам. Визначивши категорії задач, які будуть розв'язуватись певним шаром, можна вибрати деякий компонент з існуючого набору і, таким чином, створити каркас архітектури. А оскільки для кожної категорії розроблено, як правило, декілька компонентів, то можна створити множину альтернативних архітектур.

Для рішення цієї задачі було створено програмний продукт, який дозволить швидко створювати каркаси архітектур і набор типових патернів, а також добавляти свої архітектури і патерни за бажанням. Всі ці дані будуть зберігатися в структурованій базі даних(репозиторії).

Це рішення має два цілком очевидні напрямки використання.

В практичних цілях використання в фірмах по розробці програмного забезпечення для генерації на базі репозиторію великої кількості архітектурних рішення і подальшої оцінки серед генерованих архітектур з метою вибору самої оптимальної.

В навчальних цілях тобто для використання в учбових закладах для демонстрації особливостей програних архітектур, а також як можна по різному їх реалізувати чи зробити щоб студенти могли наочно оцінити різні архітектурні рішення однієї і тієї ж задачі в залежності від поставлених критеріїв. Також можна побачити реалізацію класичних патернів які використовуються в розробці програмного забезпечення.

Тобто програмний продукт, що створено, являє собою набір інструментів для створення репозиторію патернів і каркасів архітектур, а також управління цим репозиторієм, а саме редагування, додавання нових, видалення, створення

категорій патернів, а також їх категоризація за різними критеріями, також категоризація відноситься і до каркасів.

В основі стоїть каркас який включає різні шари кожен шар включає в себе набір модулів, а сама в модулі ми використовуємо конкретний патерн, для реалізації цього модуля, також необхідна система для наступного редагування вже генерованих архітектур, а конкретно для створення зав’язків між модулями і шарами, також можливість редагувати самі архітектури, дублювати добавляти видаляти створювати нові архітектури.

РОЗДІЛ 1
ОГЛЯД АРХІТЕКТУР ПРОГРАМНИХ ДОДАТКІВ ТА ПРИНЦИПИ СТВОРЕННЯ РЕПОЗИТОРІЮ ПАТЕРНІВ ТА КАРКАСІВ ПРОГРАМНИХ АРХІТЕКТУР

Розробники програмного забезпечення постійно шукають зрозумілі і витончені архітектури для своїх проектів, оскільки вони полегшують отримання реалізації без помилок і дефектів, а також краще пристосовані до розширення і використання повторно.

1.1 Введення в архітектуру програм

Протягом десятиліть розробники програмного забезпечення створювали свої проекти або з нуля, або використовуючи вже накопичений досвід, якщо такий якось вдавалося придбати. В даний час інтенсивно розвивається дисципліна програмних архітектур і проектування. Тепер ми можемо говорити про архітектури високого рівня і архітектури низького рівня в термінах, зрозумілих всім професійним розробникам. Типова схема розробки програми представлена на Рисунок 1.1.

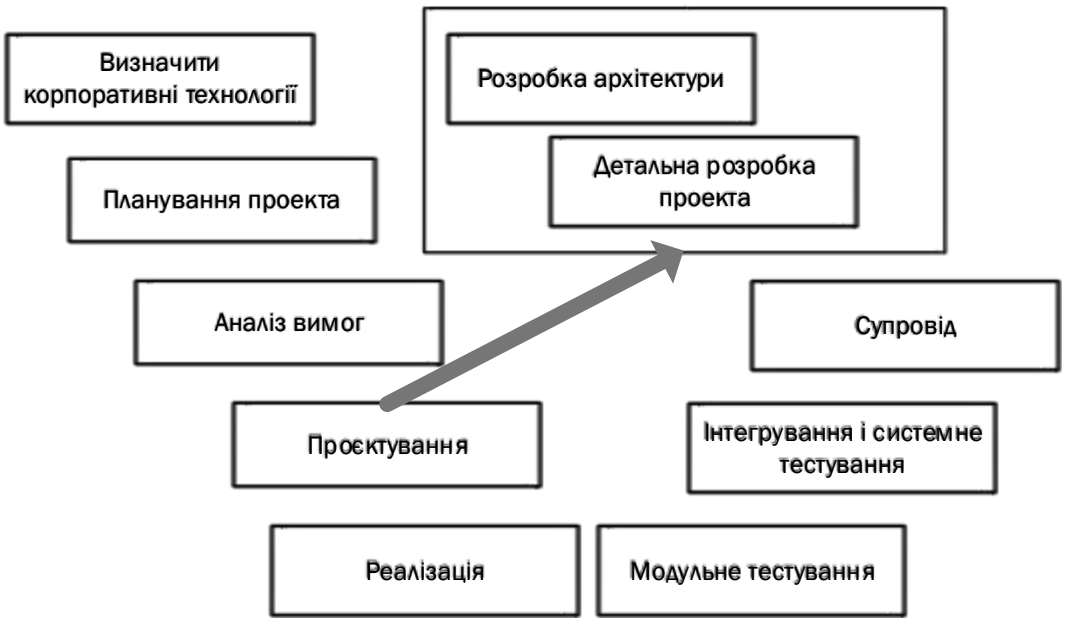


Рисунок 1.1 – Схема розробки програми

1.2 Огляд технології розробки

Будь-який додаток має апаратні і програмні компоненти. Як приклад візьмемо анти блокувальну систему автомобільних гальм. Вона має механічну, електронну і програмну складові і покликана збільшити ефективність гальмування, не допускаючи блокування коліс.

Системна розробка - це процес аналізу і проектування, який розділяє додаток на апаратні і програмні компоненти. Деякі аспекти цієї декомпозиції диктуються вимогами замовника, інші визначаються розробниками.

Процес системної розробки починається з визначення загальних системних вимог. Потім робиться вибір оптимального співвідношення між апаратним і програмним забезпеченням. Після цього визначається декомпозиція додатки на апаратне і програмне забезпечення. Потім до програмної частини застосовується технологія розробки, починаючи з аналізу вимог і т. д.

1.3 Що таке архітектура програми

Якщо порівнювати розробку програм з процесом спорудження моста, то аналіз вимог буде аналогічний вибору місць, де міст буде починатися і закінчуватися, а також визначенню роду навантажень, які він повинен витримувати. Далі, архітектор моста повинен вирішити, яким буде міст: підвісним, консольним або якогось іншого типу, що задовольняє вимогам. Іншими словами, він повинен буде визначити архітектуру моста. Розробники програмного забезпечення стикаються з схожим вибором.

Створення архітектури - це проектування на найвищому рівні. Частину процесу проектування ми називатимемо детальним проектуванням.

Ясна опис архітектури дуже важливо для всіх додатків і обов'язково в тому випадку, коли до розробки залучається велика кількість людей. Причиною цього служить необхідність розбиття всього програми на частини (модулі) з їх наступною збіркою. Вибір архітектури забезпечує необхідну модульність.

Розробники, яким доручається створення архітектури (технічні архітектори), звичайно є найбільш досвідченими в команді розробки.

1.4 Мета вибору архітектури

Для конкретного проекту розробки програмного забезпечення може бути декілька відповідних архітектур, з яких необхідно вибрати кращу. Зазвичай буває складно задовольнити всі вимоги, оскільки архітектура може виконувати одну з вимог і не виконувати іншу. З цієї причини всім вимогам необхідно присвоїти пріоритети. Наведемо приклад списку основних цілей розробки.

- Розширення.
  - Полегшення додавання нових властивостей.
- Зміни.
  - Полегшення зміни вимог.
- Простота:
  - простота розуміння;
  - простота реалізації.
- Ефективність:
  - досягнення високої швидкості: виконання і (або) компіляції;
  - досягнення малого розміру: об'єктного коду та (або) вихідного коду.

Розширення визначає ступінь, в якій архітектура повинна підтримувати додавання нових можливостей в додаток. Найчастіше чим краще архітектура пристосована до розширення, тим більш складну структуру вона має і більше часу потрібно на розробку. Розширюваність зазвичай вимагає запровадження вищих абстракцій в процес. Універсальність дає безліч переваг, але її реалізація вимагає великих витрат часу. Одним із важливих завдань при виборі ступеня універсальності є визначення класу можливих розширень. Ми не можемо проектувати в розрахунку на всі можливі розширення. У зв'язку з цим дуже корисні необов'язкові і бажані вимоги, оскільки вони показують, в який бік буде направлено розвиток програми.

Розробка з розрахунком на зміни переслідує інші цілі, хоча увазі застосування тих же прийомів проектування, що і для забезпечення розширюваності. У даному випадку ми хочемо спроектувати архітектуру таким чином, щоб вона допускала зміну вимог, наприклад щоб вимога «гравець повинен постійно мати повний контроль над своїм персонажем» можна було замінити вимогою «гравець час від часу випадковим чином втрачає контроль над своїм персонажем».

Простота є метою проектування за будь-яких обставин. Проста архітектура, яка допускає розширення і зміни, є рідкістю, і її створення вимагає великих зусиль. До інших критеріїв, використовуваним при виборі архітектури, відносяться економія машинного часу і економія пам'яті.

1.5 Огляд архетипів додатків

Розглянемо основні базові типи програмних додатків.

– Мобільні додатки. Додатки цього типу можуть розроблятися як тонкий клієнт або насичене клієнтську програму. Насичені клієнтські мобільні додатки можуть підтримувати сценарії без постійного підключення або без підключення взагалі. Веб-додатки або тонкі клієнтські програми підтримують тільки сценарії з підключенням. Обмеженням при розробці мобільних додатків можуть бути пристрої, на яких їх передбачається виконувати.

– Насичені клієнтські програми. Додатки цього типу зазвичай розробляються як самодостатні додатки з графічним інтерфейсом, який забезпечує відображення даних за допомогою набору елементів управління. Насичені клієнтські програми можуть підтримувати сценарії без підключення або без постійного підключення, якщо повинні виконувати доступ до віддалених даних або функціональності.

– Насичені Інтернет-додатки. Додатки цього типу можуть підтримувати безліч платформ і браузерів. Насичені Інтернет-додатки виконуються в ізольованій програмному середовищі браузера, яка обмежує доступ до деяких можливостям клієнта.

– Сервісні програми. Сервіси надають бізнес-функціональність для спільного використання та дозволяють клієнтам доступ до неї з локальної або віддаленої системи. Виклик операцій сервісу здійснюється за допомогою повідомлень, відповідних XML-схемам і переданих по транспортним каналам. Метою даного типу додатків є забезпечення слабкої залежності між клієнтом і сервером.

– Веб-додатки. Додатки цього типу, як правило, підтримують сценарії з постійним підключенням і різні браузери, що виконуються в найрізноманітніших операційних системах і на різних платформах.

– Існує багато інших більш спеціалізованих типів додатків. Як правило, ці типи є спеціалізаціями або поєднаннями базових типів, перерахованих у цьому списку. І тому вони не будуть розглянуті.

Мобільні додатки

Переваги

- Підтримка портативних пристроїв.
- Доступність і простота використання для мобільних користувачів.
- Підтримка сценаріїв без підключення і сценаріїв без постійного підключення.

Недоліки

- Обмежені можливості введення і навігації.
- Обмежена область відображення екрана.

Насичені клієнтські програми

Переваги

- Можливість використання ресурсів клієнта.
- Кращий час відгуку, насичена функціональність UI і покращене взаємодія з користувачем.
- Дуже динамічний взаємодія з коротким часом відгуку.
- Підтримка сценаріїв без підключення і сценаріїв без постійного підключення.

Недоліки

• Складність розгортання; при цьому широкий вибір варіантів установки, таких як ClickOnce, Windows Installer і XCOPY.

• Складнощі забезпечення сумісності версій.

• Залежність від платформи.

Насичені Інтернет- додатки (RIA)

Переваги

• Такі ж насичені можливості для користувача інтерфейсу, як і для насичених клієнтів.

• Підтримка насичених і потокових мультимедіа і графіки.

• Простота розгортання з можливостями розподілу (насиченими) такими ж, як і для Веб-клієнтів.

• Простота оновлення та зміни версій.

• Підтримка різних платформ і браузерів.

Недоліки

• Більший обсяг пам'яті, займаний на клієнті, у порівнянні з Веб-додатком.

• Обмежене використання ресурсів клієнта порівняно з насиченим клієнтським додатком.

• Необхідність розгортання на клієнті підходящої середовища виконання.

Сервісні додатки

Переваги

• Слабо пов'язане взаємодія між клієнтом і сервером.

• Можуть використовуватися різними і невзаємозв’язаними додатками.

• Підтримка для забезпечення можливості взаємодії.

Недоліки

• Відсутність підтримки UI.

• Залежність від можливості мережевого підключення.

Веб-додатки

Переваги

- Широко доступний і заснований на стандартах UI, підтримуваний на багатьох платформах.
- Простота розгортання і внесення змін.

Недоліки

- Необхідність стійкого мережевого підключення.
- Складно забезпечити насичений користувальницький інтерфейс.

Кожен тип програми може бути реалізований з використанням однієї або більше технологій. Вибір технології визначатися сценаріями і обмеженнями технологій, а також можливостями і досвідом групи розробки.

1.6 Вибір базових типів додатків

С точки зору поширеності з нині доступних 5 наведених структур програм самими універсальними і широко використовуваними являються:

1. Мобільні Програми - покривають сегмент мобільних пристроїв (планшети, телефони),
2. Насичені клієнтські Програми - покривають сегмент стаціонарних і портативних комп'ютерів і пристроїв (комп'ютери, телевізори і т.д.),
3. Сервісні додатки - надають сервіси, використовуючи які можна будувати інші додатки-клієнти.
4. Веб-Програми - покривають сегмент всіляких веб технологій.

Тому саме їх ми будемо розташовувати в репозитарії патернів та каркасів програмних архітектур.

1.7 Рекомендації з проектування багатошарових додатків

Розбиття на шари виконується відповідно логічному поділу компонентів і функціональності і не враховує фізичного розміщення компонентів. Шари можуть розміщуватися як на різних рівнях, так і на одному.

Важливо розуміти різницю між шарами і рівнями. Шари (Layers) описують логічну угруповання функцій і компонентів в додатку, тоді як рівні (tiers)

описують фізичний розподіл функцій і компонентів по серверам, комп'ютерам, мережах або віддаленим комп’ютерах. Незважаючи на те, що і для шарів, і для рівнів застосовується одна і так же термінологія (представлення, бізнес, сервіси та дані), слід пам'ятати, що тільки рівні мають на увазі фізичне розділення. Розміщення декількох шарів на одному комп'ютері (одному рівні) - досить звичайне явище. Термін рівень використовується в застосуванні до схем фізичного розподілу, наприклад, дворівневе, трирівнева, n-рівнева.

Логічний поділ на шари.

Незалежно від типу проектованого додатку і того, чи є у нього призначений для користувача інтерфейс або він є сервісним додатком, який просто надає сервіси (не плутайте з шаром сервісі), його структуру можна розкласти на логічні групи програмних компонентів. Ці логічні групи називаються шарами. Шари допомагають розділити різні типи завдань, які здійснюються цими компонентами, що спрощує створення дизайну, що підтримує можливість повторного використання компонентів. Кожен логічний шар включає ряд окремих типів компонентів, згрупованих в підшари, кожен з підшарів виконує певний тип завдань.

Визначаючи універсальні типи компонентів, які присутні в більшості рішень, можна створити схему програми або сервісу і потім використовувати цю схему як ескіз нового дизайну. Поділ програми на шари, що виконують різні ролі і функції, допомагає максимально підвищити зручність і простоту обслуговування коду, оптимізувати роботу програми при різних схемах розгортання і забезпечує чітке розмежування областей застосування певної технології або прийняття певних проектних рішень.

Шар представлення, бізнес-шар і шар даних

На найвищому і найбільш абстрактному рівні логічне уявлення архітектури системи може розглядатися як набір взаємодіючих компонентів, згрупованих в шари. На Рисунок 1.2 показано спрощене високорівневе представлення цих шарів і їх взаємовідносин з користувачами, іншими додатками, що викликають сервіси, реалізовані в бізнес-шарі додатку, джерелами даних, такими як реляційні бази даних або Веб-сервіси, що

забезпечують доступ до даних, і зовнішніми або віддаленими сервісами, використовуваними додатком.

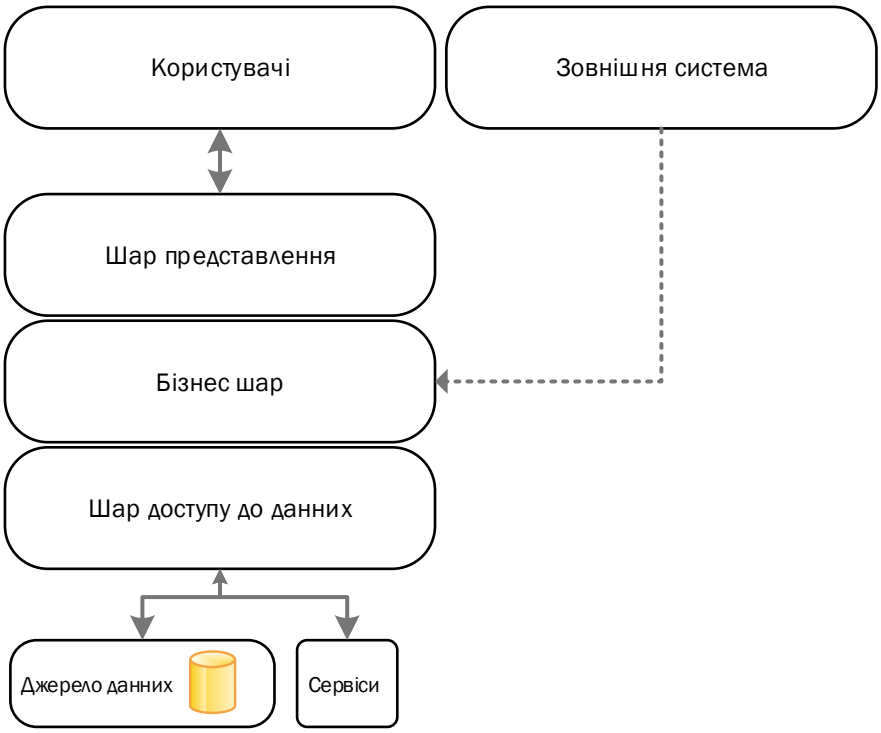


Рисунок 1.2 – Логічне уявлення архітектури багатошарової системи

Ці шари фізично можуть розташовуватися на одному або різних рівнях. Якщо вони розміщуються на різних рівнях або розділені фізичними кордонами, дизайн повинен забезпечувати це.

Додаток може складатися з ряду базових шарів.

Шар представлення. Даний шар містить орієнтовану на користувача функціональність, яка відповідає за реалізацію взаємодією користувача з системою, і, як правило, включає компоненти, що забезпечують загальну зв'язок з основною бізнес-логікою, інкапсульованною в бізнес-шарі.

Бізнес-шар. Цей шар реалізує основну функціональність системи і інкапсулює пов'язану з нею бізнес-логіку. Зазвичай він складається з компонентів, деякі з яких надають інтерфейси сервісів, доступні для використання іншими учасниками взаємодії.

Шар доступу до даних. Цей шар забезпечує доступ до даних, що зберігаються в рамках системи, і даними, що надаються іншими мережевими системами. Доступ може здійснюватися через сервіси. Шар даних надає

універсальні інтерфейси, які можуть використовуватися компонентами бізнес-шару.

Сервіси і шари.

У першому наближенні рішення, засноване на сервісах, можна розглядати як набір сервісів, що взаємодіють один з одним шляхом передачі повідомлень. Концептуально ці сервіси можна вважати компонентами рішення в цілому. Однак кожен сервіс утворений програмними компонентами, як будь-який інший додаток, і ці компоненти можуть бути логічно згруповані в шар уявлення, бізнес-шар і шар даних. Інші додатки можуть використовувати сервіси, не замислюючись про спосіб їх реалізації.

Шар сервісів.

Звичайним підходом при створенні програми, яка повинна забезпечувати сервіси для інших додатків, а також реалізовувати безпосередню підтримку клієнтів, є використання шару сервісів, який надає доступ до бізнес-функціональності додатку. Шар сервісів забезпечує альтернативне уявлення, що дозволяє клієнтам використовувати інший механізм для доступу до додатка.

У даному сценарії користувачі можуть виконувати доступ до додатка через шар уявлення, що обмінюється даними з компонентами бізнес-шару або безпосередньо, або через фасад додатки в бізнес-шарі, якщо методи зв'язку вимагають композиції функціональності. Тим часом, зовнішні клієнти та інші системи можуть виконувати доступ до додатка і використовувати його функціональність шляхом взаємодії з бізнес-шаром через інтерфейси сервісів. Це покращує можливості додатка для підтримки безлічі типів клієнтів, сприяє повторному використанню і більш високого рівня композиції функціональності в додатках.

У деяких випадках шар уявлення може взаємодіяти з бізнес-шаром через шар сервісів. Але це не є обов'язковою умовою. Якщо фізично шар уявлення(представлення) і бізнес-шар розташовуються на одному рівні, вони можуть взаємодіяти безпосередньо.

Етапи проектування багатошарової структури.

Приступаючи до проектування програми, перш за все, зосередьтеся на найвищому рівні абстракції і починайте з угруповання функціональності в шари. далі слід визначити відкритий інтерфейс для кожного шару, який залежить від типу створюваного додатка. Визначивши шари і інтерфейси, необхідно прийняти рішення про те, як буде розгортатися додаток. Нарешті, вибираються протоколи зв'язку для забезпечення взаємодії між шарами і рівнями додатка. Незважаючи на те, що розробляється структура і інтерфейси можуть змінюватися з часом, особливо в разі застосування гнучкої розробки, проходження цих етапів гарантовано забезпечить розгляд всіх важливих аспектів на початку процесу. Зазвичай при проектуванні використовується наступна послідовність кроків:

Крок 1 - Вибір стратегії поділу на шари.

Поділ на шари представляє логічне розподіл компонентів додатка по групах, які виконують певні ролі і функції. Використання багатошарового підходу може підвищити зручність послуг у програмі і спростити його масштабування, якщо необхідно підвищити продуктивність. Існує безліч різних способів групування взаємозалежних функцій в шари. Однак неправильний поділ на шари (занадто мало або занадто багато) може лише ускладнити додаток, що призводить до зниження загальної продуктивності, зручності обслуговування і гнучкості. Визначення відповідного рівня деталізації при поділі програми на шари - критично важливий перший крок у визначенні стратегії поділу на шари.

Також слід врахувати, чи застосовуються шари виключно для логічного поділу або для забезпечення фізичного поділу в разі потреби. Перетин кордонів шарів, особливо кордонів між фізично віддаленими компонентами, обумовлює виникнення витрат і зниження продуктивності. Однак загальне підвищення масштабованості і гнучкості програми може бути набагато сильнішим аргументом в порівнянні з падінням продуктивності. Крім того, поділ на шари може спростити оптимізацію продуктивності окремих шарів без впливу на суміжні шари.

У разі логічного поділу взаємодіючі шари додатки будуть розгортатися на одному рівні і виконуватися в одному процесі, що дозволить застосовувати більш продуктивні механізми зв'язку, такі як прямі виклики через інтерфейси

компонентів. Однак щоб скористатися перевагами логічного поділу на шари і гарантувати гнучкість в майбутньому, слід серйозно і ретельно підійти до питань забезпечення інкапсуляції і слабкого зв'язування між шарами.

Для шарів, розгорнутих на різних рівнях (різні фізично комп'ютери), взаємодія із суміжними шарами буде відбуватися по мережі і необхідно забезпечити, щоб вибраний дизайн підтримував відповідний механізм зв'язку, який буде враховувати затримку зв'язку та забезпечувати слабке зв'язування між шарами.

Визначення того, які частини додатка найімовірніше будуть розгортатися на різних рівнях, а які на одному, також є важливою частина стратегії поділу на шари. Для забезпечення гнучкості необхідно гарантовано забезпечити слабку зв'язаність шарів. Це дозволяє використовувати переваги більшої продуктивності при розміщенні чатин на одному рівні і в разі необхідності розгортати їх на безлічі рівнів.

Застосування багатошарового підходу може дещо ускладнити дизайн і збільшити тривалість підготовчого етапу розробки, але в разі правильної реалізації істотно поліпшить підтримку, розширюваність і гнучкість програми. Порівняйте переваги, що забезпечуються можливістю повторного використання і слабким зв'язуванням при поділі на шари, з негативними наслідками їх застосування, такими як зниження продуктивності і підвищення складності. Ретельно продумайте, як розділити додаток на шари, і як шари будуть взаємодіяти один з одним; тим самим ви забезпечите хороший баланс продуктивності і гнучкості. Як правило, виграш в гнучкості і зручності обслуговування, що забезпечується багатошарової схемою, набагато перевищує сумнівне підвищення продуктивності, якого можна досягти в тісно пов'язаному дизайні, що не використовує шари.

Крок 2 - Вибір необхідних шарів.

Існує безліч різних способів угруповання взаємозалежних функцій в шари. Найпоширеніший в бізнес-додатках підхід - розподіл функціональності уявлення, сервісів, доступу до даних і бізнес-функціональності по різним

частинам. У деяких додатках також використовуються шари складання звітів, управління та інфраструктури.

Уважно підходите до питання введення додаткових шарів. Шар повинен забезпечувати логічну угруповання взаємопов'язаних компонентів, яка помітно збільшує зручність обслуговування, масштабованість і гнучкість програми. Наприклад, якщо програма не надає сервіси, можливо, окремий шар сервісів не знадобиться, тоді додаток буде включати тільки шар уявлення, бізнес-шар і шар доступу до даних.

Крок 3 - Прийняття рішення про розподіл шарів і компонентів.

Шари і компоненти повинні розподілятися за різними фізичними рівнями, тільки якщо в цьому є необхідність. До типових причин реалізації розподіленого розгортання ставляться політики безпеки, фізичні обмеження, спільно використовувана бізнес-логіка і масштабованість.

Якщо компоненти уявлення Веб-додатки здійснюють синхронний доступ до компонентів бізнес-шару і обмеження безпеки не вимагають наявності кордону довіри між шарами, розгляньте можливість розгортання компонентів бізнес-шару і шару уявлення на одному рівні, це забезпечить максимальну продуктивність і керованість.

У насичених клієнтських додатках, в яких обробка UI виконується на клієнтському комп'ютері, варіант розгортання компонентів бізнес-шару на окремому бізнес-рівні може бути обраний по міркуваннями безпеки і для підвищення керованості.

Розгортається бізнес-суть на одному рівні з кодом, що її використовують. Це може означати їх розгортання в декількох місцях, наприклад, розміщення копій на окремому рівні уявлення або даних, логіка якого використовує або посилається на ці бізнес-суті. Розгортається компоненти агентів сервісу на тому ж рівні, що і код, що викликає ці компоненти, якщо обмеження безпеки не вимагають наявності кордону довіри між ними.

Розгляньте можливість розгортання асинхронних компонентів бізнес-шару, компонентів робочого процесу і сервісів з однаковими характеристиками завантаження і введення / виводу на окремому рівні. Це дозволить

налаштовувати інфраструктуру для забезпечення максимальної продуктивності і масштабованості.

Крок 4 - З'ясування можливості згортання шарів.

У деяких випадках має сенс звернути шари. Наприклад, в додатку, що має дуже обмежений набір бізнес-правил або використовує правила переважно для валідації, бізнес-логіка і логіка уявлення можуть бути реалізовані в одному шарі. У додатку, який просто отримує дані з Веб-сервісу і відображає їх, може мати сенс просто додати посилання на Веб-сервіс безпосередньо в шар уявлення і використовувати дані Веб-сервісу безпосередньо. В цьому випадку логічно об'єднуються шари доступу до даних та представлення.

Це лише деякі приклади того, коли має сенс згортання шарів. Проте, угрупування функціональності в шари є загальним правилом. У деяких випадках шар може виступати в ролі проксі або транзитного рівня, який забезпечує інкапсуляцію або слабке зв'язування практично без надання функціональності. Але відділення цієї функціональності дозволить розширювати її в майбутньому без здійснення впливу або з невеликим впливом на інші частини.

Крок 5 - Визначення правил взаємодії між шарами.

Коли справа доходить до стратегії поділу на шари, необхідно визначити правила взаємодії шарів один з одним. Основна мета завдання правил взаємодії - мінімізація залежностей і виключення циклічних посилань. Наприклад, якщо два шари мають залежності від компонентів третього шару, з'являється циклічна залежність. Загальним правилом, якого слід дотримуватися в даному випадку, є дозвіл тільки односпрямованого взаємодії між шарами через застосування однієї з таких взаємодій:

• Взаємодія зверху вниз. Шари можуть взаємодіяти з шарами, розташованими нижче, але нижні шари ніколи не можуть взаємодіяти з розташованими вище шарами. Це правило допоможе уникнути циклічних залежностей між шарами. Використання подій дозволить сповіщати компоненти розташованих вище шарів про зміни в нижніх шарах без введення залежностей.

• Суворе взаємодія. Кожен шар повинен взаємодіяти тільки із шаром, розташованим безпосередньо під ним. Це правило забезпечить суворий поділ,

при якому кожен шар знає тільки про шар відразу під ним. Позитивний ефект від цього правила в тому, що зміни в інтерфейсі шару будуть впливати тільки на шар, розташований безпосередньо над ним. Застосовуйте цей підхід при проектуванні програми, яке передбачається розширювати новою функціональністю в майбутньому, якщо хочете максимально скоротити вплив цих змін; або при проектуванні програми, для якої необхідно забезпечити можливість розподілу на різні рівні.

• Вільна взаємодія. Більш високі шари можуть взаємодіяти з розташованими нижче шарами безпосередньо, в обхід інших верств. Це може підвищити продуктивність, але також збільшить залежності. Інакше кажучи, зміни в нижньому шарі може впливати на кілька розташованих вище шарів. Цей підхід рекомендується застосовувати при проектуванні програми, яка гарантовано буде розміщуватися на одному рівні (наприклад, самодостатнє насичене клієнтську програму), або при проектуванні невеликого додатка, для якого внесення змін, які зачіпають безліч шарів, не потребує великих зусиль.

Крок 6 - Визначення наскрізної функціональності.

Визначившись з шарами, необхідно звернути увагу на функціональність, що охоплює всі шари. Таку функціональність часто називають наскрізною функціональністю. До неї відноситься протоколювання, валідація, аутентифікація і управління винятками. Важливо виявити всі наскрізні функції програми і по можливості спроектувати для кожної з них окремі компоненти. Такий підхід допоможе забезпечити кращу можливість повторного використання і обслуговування.

Уникайте змішування такого загального коду з кодом компонентів шарів, щоб шари і їх компоненти викликали компоненти наскрізної функціональності тільки для виконання таких дій, як протоколювання, кешування або аутентифікація. Оскільки ця функціональність повинна бути доступна для всіх верств, спосіб розгортання компонентів наскрізний функціональності повинен забезпечувати це, навіть якщо шари фізично розміщуються на різних рівнях.

Крок 7 - Визначення інтерфейсів між шарами.

Основна мета при визначенні інтерфейсу шару - забезпечити слабке зв'язування між шарами. Це означає, що шар не повинен розкривати внутрішні деталі, від яких може залежати інший шар. Замість цього інтерфейс шару повинен бути спроектований так, щоб звести до мінімуму залежності шляхом надання відкритого інтерфейсу, що приховує деталі компонентів шару. Таке приховування називається абстракцією. Існує безліч способів реалізувати її. Пропонуємо підходи, які можуть використовуватися для визначення інтерфейсу шару:

• Абстрактний інтерфейс. Може бути визначений за допомогою абстрактного базового класу або інтерфейсу, який виступає в ролі опису типу для конкретних класів. Цей тип визначає загальний інтерфейс, використовуваний для взаємодії з цим шаром. Такий підхід також покращує тетстування, тому що дозволяє використовувати тестові об'єкти (іноді їх називають mock-об'єктами або фіктивними об'єктами), що реалізують абстрактний інтерфейс.

• Загальний тип проектування. Багато шаблонів проектування визначають конкретні типи об'єктів, які представляють інтерфейс в різних шарах. Ці типи об'єктів забезпечують абстракцію, яка приховує деталі, що стосуються шару. Наприклад, шаблон Table Data Gateway визначає типи об'єктів, які представляють таблиці в базі даних і відповідають за реалізацію SQL-запитів, необхідних для доступу до даних. Суті, працюють з об'єктом, нічого не знають про SQL-запити або деталях того, як об'єкт підключається до бази даних і виконує команди. Багато шаблонів проектування базуються на абстрактних інтерфейсів, але в основі деяких з них лежать конкретні класи. Більшість шаблонів, такі як Table Data Gateway, добре задокументовані в цьому відношенні. Загальні типи проектування слід застосовувати, якщо необхідний спосіб швидко і просто реалізувати інтерфейс шару або при реалізації шаблону проектування для інтерфейсу шару.

• Інверсія залежностей. Це такий стиль програмування, при якому абстрактні інтерфейси визначаються поза або незалежно від шарів. Тоді шари залежать не друг від друга, а від загальних інтерфейсів. Шаблон Dependency

Injection є типовою реалізацією інверсії залежностей. При використанні Dependency Injection контейнер описує зіставлення, що визначають як знаходити компоненти, від яких можуть залежати інші компоненти, і контейнер може створювати і запроваджувати ці залежні компоненти автоматично. Підхід з інверсією залежностей забезпечує гнучкість і може допомогти в реалізації модульного дизайну, оскільки залежно від конфігурації, а не кодом. Також такий підхід максимально спрощує тестування, тому що дозволяє вводити тестові класи на різні рівні дизайну.

• Заснований на обміні повідомленнями. Замість взаємодії з компонентами інших верств безпосередньо через виклик їх методів або доступ до властивостей можна використовувати зв'язок за допомогою обміну повідомленнями для реалізації інтерфейсів і забезпечення взаємодії між шарами. Існує кілька рішень для обміну повідомленнями, такі як Windows Communication Foundation, Веб-сервіси і Microsoft Message Queuing, які підтримують взаємодію через фізичні кордони і кордону процесів. Можна також комбінувати абстрактні інтерфейси із загальним типом повідомлень, що використовуються для визначення структур даних для взаємодії. Основна відмінність підходу на основі повідомлень в тому, що для взаємодії між шарами використовується загальний інтерфейс, що інкапсулює всі деталі взаємодії. Цей інтерфейс може визначати операції, схеми даних, контракти повідомлення про збої, політики системи безпеки і багато інших аспектів, які стосуються обміну даними між шарами. Заснований на обміні повідомленнями підхід рекомендується застосовувати при реалізації Веб-додатки та описі інтерфейсу між шаром уявлення і бізнес-шаром, який повинен підтримувати безліч типів клієнтів, або якщо потрібно підтримувати взаємодію через фізичний кордон і кордону процесів. Також розгляньте можливість застосування такого підходу, якщо хочете формалізувати взаємодію або взаємодіяти з інтерфейсом, що не зберігає стан, коли дані про стан передаються з повідомленням.

Для реалізації взаємодії між шаром уявлення Веб-додатки та шаром бізнес-логіки рекомендується використовувати підхід на основі повідомлень. Якщо бізнес-шар не зберігає стану між викликами (іншими словами, кожен виклик між

шаром уявлення і бізнес-шаром представляє новий контекст), можна передавати дані контексту разом із запитом і забезпечити загальну модель обробки виключень і помилок в шарі уявлення.

Крок 8 - Вибір стратегії розгортання.

Існує кілька загальних шаблонів, які представляють структури розгортання додатків, що застосовуються в багатьох рішеннях. Там, де необхідно вибрати найбільш відповідне рішення розгортання для додатка, корисно спочатку розглянути загальні шаблони. Тільки повністю розібравшись з різними схемами розгортання, можна переходити до конкретних сценаріями, вимогам і обмеженням безпеки, щоб визначитися з найбільш підходящим шаблоном або шаблонами.

Крок 9 - Вибір протоколів зв'язку.

Фізичні протоколи, використовувані для зв'язку між шарами або рівнями, грають основну роль в забезпеченні продуктивності, безпеки і надійності додатки. Вибір протоколу зв'язку має ще більше значення для розподіленого розгортання. Якщо компоненти розміщуються фізично на одному рівні, часто можна покластися на пряму взаємодію цих компонентів. Але якщо компоненти і шари розгорнуті фізично на різних серверах і клієнтських комп'ютерах, як це відбувається в більшості сценаріїв, необхідно продумати, як забезпечити ефективну і надійну зв'язок між компонентами цих шарів.

1.7.1 Веб-додаток

Веб-додаток - це програма, яка може використовуватися користувачами через веб-браузер або спеціалізований агент користувача. Браузер створює HTTP-запити до певних URL, які зіставляються з ресурсами на Веб-сервері. Сервер формує візуальне уявлення HTML-сторінок, яке може бути відображено браузером, і повертає їх клієнту. Ядро Веб-додатка - це його логіка на стороні сервера. Такий додаток може складатися з безлічі окремих шарів. Типовим прикладом є тришарова архітектура, що включає шар уявлення, бізнес-шар і шар доступу до даних. Рисунок 1.3 ілюструє типову архітектуру Веб-додатка з

найчастіше вживаними компонентами, згрупованими по функціональних областях.

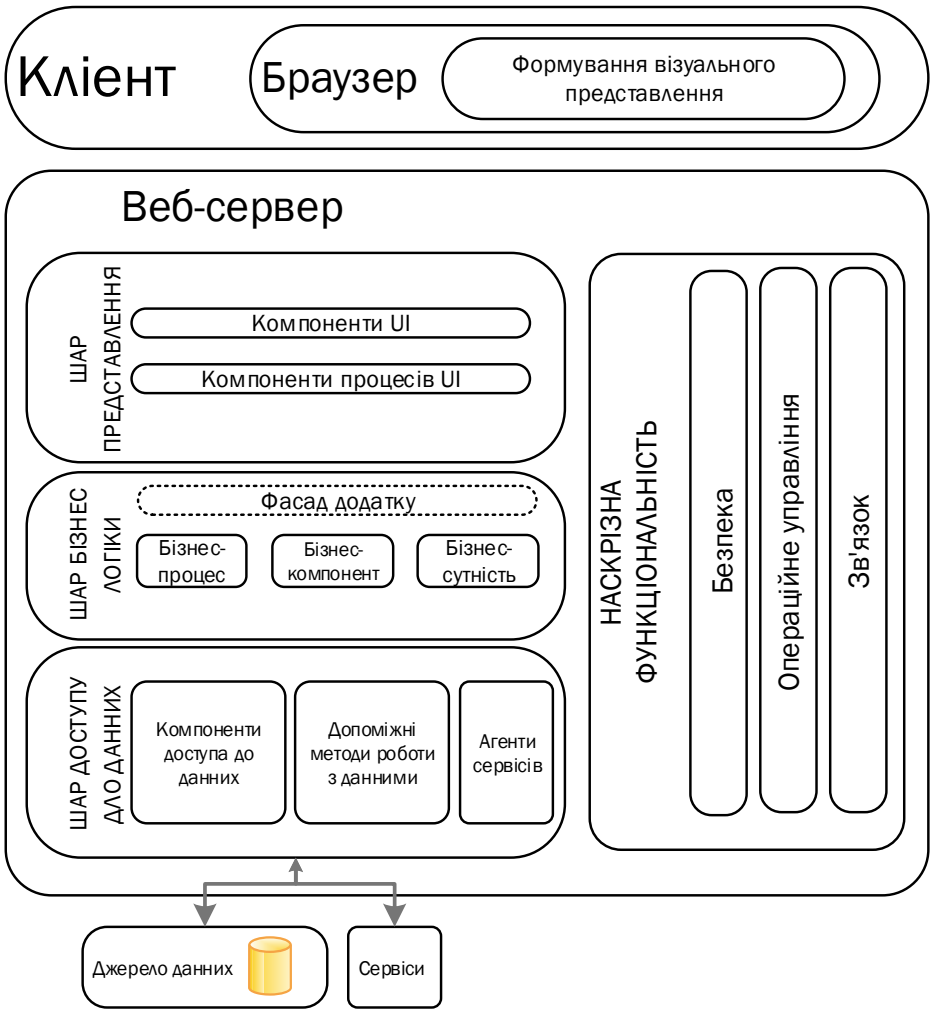


Рисунок 1.3 – Типова структура Веб-додатка

Шар уявлення зазвичай включає компоненти UI і логіки подання; бізнес-шар - компоненти бізнес-логіки, бізнес-процесу і бізнес-сутностей, а також іноді фасад; шар доступу до даних - компоненти доступу до даних і агенти сервісів.

1.7.2 Проектування насичених клієнтських додатків

Користувальницькі інтерфейси насичених клієнтів можуть забезпечувати високу продуктивність, інтерактивність і насичене взаємодія з користувачем для додатків, які повинні працювати як самодостатні програми та в сценаріях з підключенням, без постійного підключення і без підключення. Windows Forms, Windows Presentation Foundation (WPF) і Microsoft Office Business Application

(ОВА) - все це доступні середовища та інструменти розробки, що дозволяють розробникам швидко і без праці створювати насичені клієнтські програми.

Ці технології можуть використовуватися для створення як самодостатніх додатків, так і додатків, що виконуються на клієнтському комп'ютері, але взаємодіючих з сервісами, що надаються іншими рівнями (як логічними, так і фізичними), і іншими додатками, що надають операції, які необхідні клієнту. До цих операцій належать доступ до даних, вилучення даних, пошук, відправка даних в інші системи, резервне копіювання і т.д. На Рисунок 1.4, представлений загальний вид архітектури типового насиченого клієнта і показані компоненти, зазвичай розташовуються в кожному з шарів.

Рисунок 1.4 – Загальне уявлення архітектури типового насиченого клієнтського додатка

Типове насичене клієнтське додатка включає три шари: шар представлення, бізнес-шар і шар доступу до даних. Шар представлення, як правило, містить компоненти UI і логіки подання; бізнес-шар - компоненти бізнес-логіки, бізнес-процесу і бізнес-сутностей; і шар доступу до даних - компоненти доступу до даних і агентів сервісів.

Насичені клієнтські програми можуть бути досить тонкими додатками, що складаються, головним чином, з шару уявлення, який за допомогою сервісів виконує доступ до віддаленого бізнес-рівню, розміщеному на серверах. Прикладом такого додатка є додаток для введення даних, передавальне всі дані на сервер для обробки та зберігання.

І навпаки, насичені клієнтські програми можуть бути дуже складними додатками, які здійснюють більшу частину обробки самостійно і взаємодіють з іншими сервісами та сховищами даних для отримання або відправки даних. Прикладом такого додатка є ПЗ на базі Microsoft Excel®, яке виконує складні завдання локально, зберігає стан і дані локально і взаємодіє з віддаленими серверами тільки для витягання і поновлення пов'язаних даних. Такі насичені клієнти мають власні бізнес-шари та шари доступу до даних. Рекомендації по проектуванню бізнес-шарів і прошарків доступу до даних в таких додатках аналогічні рекомендаціям для всіх інших додатків.

1.7.3 Проектування насичених Інтернет-додатків

Насичених Інтернет-додатків (Rich Internet Applications, RIA) підтримують насичені графічні елементи і сценарії із застосуванням потокового мультимедіа, забезпечуючи при цьому переваги розгортання та зручності обслуговування, властиві Веб-додаткам. RIA можуть виконуватися в підключеному модулі браузера, такому як Microsoft® Silverlight®, на відміну від розширень, що використовують код браузера, таких як Asynchronous JavaScript і XML (AJAX). Типова реалізація RIA використовує Веб-інфраструктуру в поєднанні з клієнтським додатком, відповідальним за обробку уявлення. Плагін забезпечує бібліотечні процедури для підтримки насиченої графіки, а також контейнер, в цілях безпеки обмежує доступ до локальних ресурсів. RIA можуть виконувати

більш розширений і складний код на стороні клієнта, ніж звичайне Веб-додаток, забезпечуючи тим самим можливість скоротити навантаження на Веб-сервер. На Рисунок 1.5, представлена типова структура реалізації RIA.

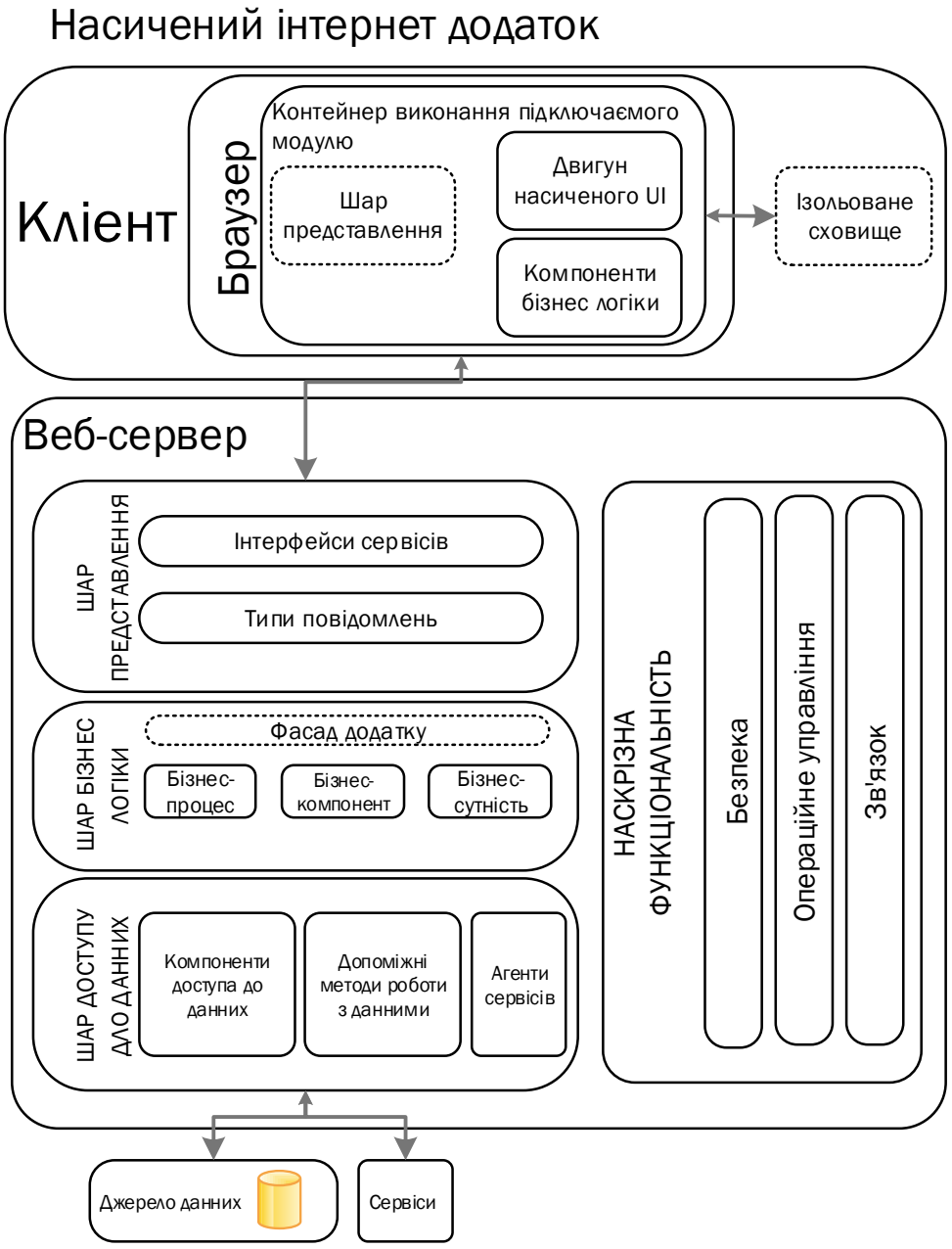


Рисунок 1.5 – Архітектура типовий реалізації RIA. Пунктиром позначені необов'язкові компоненти

Типове насичене Інтернет-додаток включає три шари: шар уявлення, бізнес-шар і шар доступу до даних. Шар представлення, як правило, містить компоненти UI і логіки подання; бізнес-шар - компоненти бізнес-логіки, бізнес-

процесу і бізнес-сутностей; і шар доступу до даних - компоненти доступу до даних і агентів сервісів.

У RIA частина бізнес-процесів і навіть код доступу до даних часто переносяться на клієнт. Тому клієнтська частина, залежно від сценарію, може включати деяку або всю функціональність бізнес-шару і шару доступу до даних. На Рисунок 1.5, показано, як деякі бізнес-процеси зазвичай реалізовуються на клієнті.

RIA можуть бути як тонкими інтерфейсами для серверних бізнес-сервісів, так і складними додатками, які самостійно виконують більшу частину процесів і взаємодіють з сервісами на сервері тільки для отримання або відправки даних. Отже, дизайн та реалізація RIA можуть бути дуже різноманітними. Проте, існує ряд загальних підходів, що забезпечують створення гарної архітектури шару вистави та її взаємодії з сервісами на сервері. Багато хто з них ґрунтуються на загальновідомих шаблонах проектування, які сприяють побудові додатка з окремих компонентів. Це забезпечує скорочення залежностей, спрощення обслуговування і тестування, а також більш широкі можливості повторного використання.

1.7.4 Проектування мобільних додатків

Як правило, звичайне мобільний додаток структурований як багатошарове додаток, що складається з шару представлення, бізнес-шару і шару доступу до даних. При розробці мобільного додатку можна створювати тонкий Веб-клієнт або насичений клієнт. Для насиченого клієнта бізнес-шар і шар сервісів даних, швидше за все, будуть розташовуватися на самому пристрої. Для тонкого клієнта всі шари будуть розміщені на сервері. Рисунок 1.6, ілюструє типову архітектуру насиченого клієнтського мобільного додатку, компоненти якого згруповані по функціональних областях.

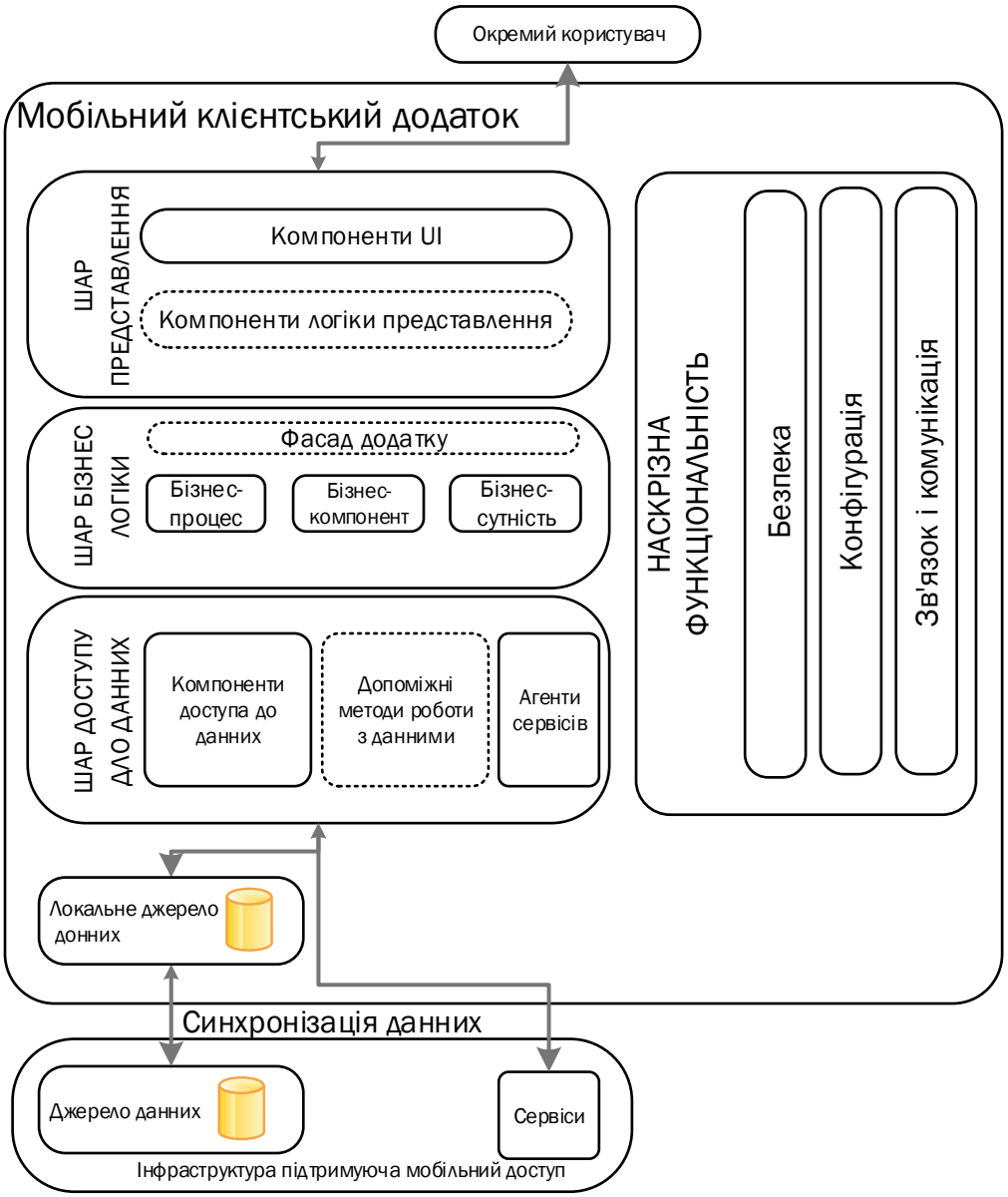


Рисунок 1.6 – Типова структура мобільного додатку

Як правило, в шарі представлення мобільного додатку розташовуються компоненти користувальницького інтерфейсу і також, можливо, компоненти логіки подання. Бізнес-шар, якщо такий є, зазвичай включає компоненти бізнес-логіки, всі компоненти бізнес-процесу і бізнес-сутностей, необхідні додатком, і фасад, якщо він використовується. У шарі доступу до даних знаходяться компоненти доступу до даних і агентів сервісів. Щоб скоротити об'єм займаної додатком пам'яті пристрою, мобільні додатки зазвичай використовують менш жорсткі підходи до поділу на шари і менше число компонентів.

1.8 Що таке патерн проектування

За словами Крістофера Олександра, «будь-який патерн описує завдання, яка знову і знову виникає в нашій роботі, а також принцип її рішення, причому таким чином, що це рішення можна потім використовувати мільйон разів, нічого не вигадуючи заново». Хоча Олександр мав на увазі патерни, що виникають при проектуванні будинків і міст, але його слова вірні і відносно патернів об'єктно-орієнтованого проектування. Наші рішення виражаються в термінах об'єктів і інтерфейсів, а не стін і дверей, але в обох випадках сенс патерну - запропонувати вирішення певної задачі в конкретному контексті.

У загальному випадку патерн складається з чотирьох основних елементів:

1. Ім'я - посилаючись на нього, ми можемо відразу описати проблему проектування, її рішення і їх наслідки. Присвоєння патернам імен дозволяє проектувати на більш високому рівні абстракції. За допомогою словника патернів можна вести обговорення з колегами, згадувати патерни в документації, в тонкощах представляти дизайн системи.

2. Завдання - опис того, коли слід застосовувати патерн. Необхідно сформулювати завдання і його контекст. Може описуватися конкретна проблема проектування, наприклад спосіб представлення алгоритмів у вигляді об'єктів. Іноді відзначається, які структури класів або об'єктів свідчать про негнучкий дизайн. Також може включатися перелік умов, при наявності яких доцільно буде застосувати даний патерн.

3. Рішення - опис елементів дизайну, відносин між ними, функцій кожного елемента. Конкретний дизайн або реалізація не маються на увазі, оскільки патерн - це шаблон, який можна застосовувати в самих різних ситуаціях. Просто дається абстрактне опис завдання проектування і того, як вона може бути вирішена за допомогою якогось вельми узагальненого поєднання елементів.

4. Результати - це наслідки застосування патерну і різного роду компроміси. Хоча при описі проектних рішень про наслідки часто не згадують, знати про них необхідно, щоб можна було вибрати між різними варіантами і

оцінити переваги і недоліки даного патерну. Тут мова йде і про вибір мови та реалізації. Оскільки в об'єктно-орієнтованому проектуванні повторне використання часто є важливим фактором, то до результатів слід відносити і вплив на ступінь гнучкості, розширюваності і переносимості системи. Перерахування всіх наслідків допоможе вам зрозуміти і оцінити їх роль.

Те, що один сприймає як патерн, для іншого просто будівельний блок. патерни проектування - це не те ж саме, що пов'язані списки або хеш-таблиці, які можна реалізувати у вигляді класу і повторно використовувати без яких би то не було модифікацій. Але це і не складні, предметно-орієнтовані рішення для цілого додатки або підсистеми. Під патернами проектування розуміється опис взаємодії об'єктів і класів, адаптованих для вирішення загальної задачі проектування в конкретному контексті.

Патерн проектування іменує, абстрагує і ідентифікує ключові аспекти структури спільного рішення, які і дозволяють застосувати його для створення повторно використовуваного дизайну. Він виокремлює класи і екземпляри, їх роль і відносини, а також функції. При описі кожного патерну увага акцентується на конкретному завданні об'єктно-орієнтованого проектування. Аналізується, коли слід застосовувати патерн, чи можна його використовувати з урахуванням інших проектних обмежень, які будуть наслідки застосування методу.

Хоча, строго кажучи, патерни використовуються в проектуванні, вони засновані на практичних рішеннях, реалізованих на основних мовах об'єктно-орієнтованого програмування типу Java і C++, а не на процедурних (Pascal, C, Basic і т.п.) або об'єктно-орієнтованих мовах з динамічною типізацією (PHP, Ruby, Perl, JavaScript).

РОЗДІЛ 2
КОНЦЕПЦІЯ ПОБУДОВИ РЕПОЗИТОРІЮ ПАТЕНРІВ ФУНКЦІОНАЛЬНИХ КОМПОНЕНТІВ ТА КАРКАСІВ ПРОГРАМНИХ АРХІТЕКТУР

2.1 Опис патернів проектування

Як виконувати опис патерннів проєктування? Графічних зображень недостатньо. Вони всього лише символізують кінцевий продукт процесу проєктування у вигляді відносин між класами і об'єктами. Щоб повторно використати дизайном, нам потрібно документувати рішення, альтернативні варіанти і компроміси, які спричинили до нього. Пріоритетні також конкретні приклади, бо вони надають можливість побачити застосування патерну.

При описі патернів проєктуванні ми будемо притримуватися єдиного принципу. Опис кожного патерну розбито на розділи, перераховані нижче. Такий підхід надає можливість одноманітно представити інформацію, полегшує вивчення, порівняння та застосування патернів.

Назва та класифікація патерна.

Назва патерну має чітко відображати його призначення.

Призначення.

Лаконічну відповідь на наступні питання: які функції патерну, його обґрунтування і призначення, яку конкретну задачу проєктування можна вирішити з його допомогою.

Відомий також під ім'ям.

Інші поширені назви патерну, коли такі є.

Мотивація.

Сценарій, який ілюструє завдання проєктування і то, як вона вирішується даною структурою класу або об'єкта. Завдяки мотивації можна краще зрозуміти наступне, більш абстрактне опис патерна.

Застосовність.

Опис ситуацій, в яких можна застосовувати даний патерн. Приклади проєктування, які можна поліпшити за його допомогою. Розпізнавання таких ситуацій.

Структура.

Графічне представлення класів в патерні з використанням нотації Object Modeling Technique (OMT). Ми користуємося також діаграмами взаємодій для ілюстрації послідовностей запитів і відносин між об'єктами. У додатку В ця нотація описується докладно.

2.2 Організація каталогу патернів

Патерни проектування повинні бути класифіковані відповідно до свого рівня абстракції та призначення. В таблиці 2.1 показано класифікацію патернів за двома ознаками. Перший з них мета природньо служить для відображення призначення патерну проектування. У відповідності до цього критерію патерни бувають породжуючі, поведінкові та структурні. Відповідно перші виконують функції створення об'єктів певних сутностей у програмі, другі реалізують певні шаблони поведінки і взаємодії класів, а треті служать для композиції класів у програмі.

Інший параметр класифікації патернів – рівень абстракції – ілюструє їх належність до рівня класів чи рівня об'єктів. Відповідно патерни рівня класів служать для реалізації взаємодії на абстрактоному рівні класів (типів) даних. А патерни на рівні об'єктів мають стосунок до реалізації взаємодії об'єктів класів.

Породжуючі патерни класів частково делегують відповідальність за створення об'єктів своїм підкласам, тоді як породжують патерни об'єктів передають відповідальність іншому об'єкту. Структурні патерни класів використовують наслідування для утворення класів, в той час як структурні патерни об'єктів описують способи компоновки об'єктів з частин. Поведінкові патерни класів використовують успадкування для опису алгоритмів і потоку управління, а поведінкові патерни об'єктів описують, як об'єкти, що належать

деякій групі, спільно функціонують і виконують завдання, яка жодному окремому об'єкту не під силу.

Існують і інші способи класифікації патернів. Деякі патерни часто використовуються разом. Наприклад, компонувальник застосовується з ітератором або відвідувачем. Деякими патернами пропонуються альтернативні рішення. Так, прототип нерідко можна використовувати замість абстрактної фабрики. Застосування частини патернів призводить до подібного дизайну, хоча спочатку їх призначення по-різному. Наприклад, структурні діаграми компонувальника і декоратора схожі.

Таблиця 2.1 – Простір патернів проєктування

Породжуючі Патерни	Factory Method (Фабричний метод)
	Abstract Factory (Абстрактна фабрика)
	Singleton (Одинак)
	Prototype (Прототип)
	Builder (Будівельник)
Структурні Патерни	Adapter (адаптер) класу
	Adapter (адаптер)об'єкта
	Decorator (Декоратор)
	Proxy (Заступник)
	Composite (Компоновщик)
	Bridge (Міст)
	Flyweight (Пристосуванець)
	Facade (Фасад)

Продовження таблиці 2.1

Паерни Поведінки	Шаблонний метод (Template Method) Iterator (Ітератор)
	Command (Команда)
	Observer (Спостерігач)
	Visitor (Відвідувач)
	Mediator (Посередник)
	State (Стан)
	Strategy (Стратегія)
	Memento (Хранитель)
	Chain of Responsibility (Ланцюжок обов'язків)
	Mapper (Распределитель)
	Money (Гроші)
	Special Case (Особливий Випадок)
	Plugin (Плагін)
	Gateway (Шлюз)
Базові патерни	Separated Interface (Виділений інтерфейс)
	Registry (Реєстр)
	Service Stub (Сервісна заглушка)
	Value Object (Об'єкт-значення)
	Record Set ()
	Layer Supertype (супертіпа Рівня)
Патерни веб-уявлення	Transform View (Перетворювач)
	Template View (шаблонизатор)
	Application Controller (Контролер додатки)
	Two Step View (двокрокового шаблонізації)
	Page Controller (Контролер сторінки)
	Front Controller (Контролер входу / Єдина точка входу)
	MVC - Model View Controller (Модель-Вид-Контролер)

Продовження таблиці 2.1

Патерни об'єктно-реляційної структурування	Identity Field (Поле первинного ключа)
	Foreign Key Mapping (Розмітка зовнішніх ключів)
	Association Table Mapping (Розмітка таблиць зв'язків)
	Dependent Mapping (Управління розподілом підлеглих сутностей)
	Embedded Value (Об'єднане властивість)
	Serialized LOB (серіалізовані LOB)
	Single Table Inheritance (Спадкування з єдиною таблицею)
	Class Table Inheritance (Спадкування з таблицями класів)
	Concrete Table Inheritance (Спадкування з таблицями кінцевих класів)
	Inherritance Mappers (Успадковані розподільники)
Патерни логіки сутності	Transaction Script (Сценнарій транзакції)
	Domain Model (Модель області визначення)
	Table Module (Дескриптор таблиці)
	Service Layer (Сервісний рівень)
Патерни локальної конкуренції	Optimistic Offline Lock (Оптимістична блокування)
	Pessimistic Offline Lock (песимістична блокування)
	Coarse Grained Lock (Груба блокування)
	Implicit Lock (Прихована блокування)

Ясно, що організувати патерни проєктування допустимо багатьма способами. Оцінюючи патерни з різних точок зору, ви глибше зрозумієте, як вони функціонують, як їх порівнювати і коли застосовувати той чи інший.

Головним чином існує три типи патерни проектування:

1. Креаційні.

Ці паттерни проектування стосуються опису класів або створення об'єкта. Ці шаблони можна додатково класифікувати на шаблони творчості класу та шаблони створення об'єктів. У той час як шаблони створення класів ефективно використовують успадкування в процесі інстанції, шаблони створення об'єктів ефективно використовують делегування, щоб виконати роботу.

Креаційні шаблони - це метод "Фабрика", "Фабрика абстрактних", "Будівельник", "Синглтон", "Об'єктний пул" та "Прототип".

1) Припустимо, розробник хоче створити простий клас DBConnection для підключення до бази даних та хоче отримати доступ до бази даних в декількох місцях з коду, як правило, розробник створить екземпляр класу DBConnection та використовувати це для виконання операцій з базою даних, де потрібно. Це призводить до створення декількох підключень до бази даних, оскільки кожен екземпляр класу DBConnection матиме окреме підключення до бази даних. Для того, щоб мати справу з цим, ми створюємо клас DBConnection як клас однотонних, так що створюється лише один екземпляр DBConnection і встановлюється єдине з'єднання. Оскільки ми можемо керувати підключенням DB через один екземпляр, щоб ми могли контролювати баланс завантаження, непотрібні з'єднання тощо.

2) Припустимо, ви хочете створити кілька екземплярів подібного роду і хочете досягти вільної зв'язку, тоді ви можете перейти до заводської схеми. Клас, що реалізує заводський шаблон проектування, працює як міст між декількома класами. Розглянемо приклад використання декількох серверів баз даних, таких як SQL Server та Oracle. Якщо ви розробляєте додаток, використовуючи базу даних SQL Server в якості зворотного кінця, але в майбутньому вам потрібно буде змінити базу даних на oracle, вам потрібно буде змінити весь код, так як фабричні шаблони проектування підтримують нещільне з'єднання та просту реалізацію, нам слід зайти на завод досягнення нещільного з’єднання та створення об'єкта подібного роду.

2. Структурні.

Ці схеми проектування стосуються організації різних класів та об'єктів для формування більших структур та надання нових функціональних можливостей.

Структурними моделями дизайну є адаптер, міст, композит, декоратор, фасад, легка вага, дані приватного класу та проксі.

1) Коли два інтерфейси не сумісні один з одним і хочуть встановити зв'язок між ними за допомогою адаптера, його називають схемою дизайну адаптера. Шаблон адаптера перетворює інтерфейс класу в інший інтерфейс або класи, які клієнт очікує, що адаптер дозволяє класам працювати разом, що не може бути інакше через несумісність. тому в таких типах несумісних сценаріїв ми можемо перейти до схеми адаптера.

3. Поведінкові.

Моделі поведінки стосуються виявлення загальних моделей комунікації між об'єктами та реалізації цих зразків.

Моделі поведінки - це ланцюжок відповідальності, команда, перекладач, ітератор, посередник, коментар, нульовий об’єкт, спостерігач, стан, стратегія, метод шаблону, відвідувач

1) Шаблон патерну визначає кістяк алгоритму в операції, що визначає деякі етапи для підкласів, метод Шаблон дозволяє підкласам переглядати певні етапи алгоритму без зміни структури алгоритму. Скажімо, для прикладу у вашому проекті ви хочете, щоб поведінку модуля можна було розширити, таким чином, щоб ми могли змусити модуль вести себе по-новому і по-різному в міру зміни вимог програми або задоволення потреб нових програм. Однак нікому не дозволяється вносити зміни до вихідного коду. це означає, що ви можете додавати, але не можете змінювати структуру в цих сценаріях, розробник може наближатися до шаблону дизайну шаблонів.

2.3 Класифікація патернів проєктування

Фабричний метод.

Фабричний метод - це креаційний шаблон проектування, тобто пов'язаний зі створенням об'єкта. У заводському шаблоні ми створюємо об'єкт, не піддаючи клієнту логіку створення, а клієнт використовує той самий загальний інтерфейс для створення нового типу об'єкта.

Ідея полягає у використанні статичної функції-члена (статичний заводський метод), який створює та повертає екземпляри, приховуючи деталі модулів класу від користувача.

Фабричний візерунок є одним із основних принципів проектування для створення об’єкта, що дозволяє клієнтам створювати об'єкти бібліотеки (пояснено нижче) таким чином, щоб у неї не було тісного зв’язку з ієрархією класів бібліотеки.

Що мається на увазі, коли ми говоримо про бібліотеку та клієнтів? Бібліотека - це те, що надається якоюсь третьою стороною, яка відкриває деякі публічні API та клієнти звертаються до цих публічних API, щоб виконати своє завдання. Дуже простим прикладом можуть бути різні види переглядів, надані ОС Android.

Як ви, мабуть, зауважили у наведеному вище прикладі, Клієнт створює об'єкти або TwoWheeler, або FourWheeler на основі деякого входу під час створення свого об'єкта. Скажімо, бібліотека представляє новий клас ThreeWheeler, який також включає три колесні транспортні засоби. Що б сталося? Клієнт в кінцевому підсумку прикуває нове інше, якщо в умовних сходах створити об'єкти ThreeWheeler. Що, в свою чергу, потребуватиме перекомпіляції Клієнта. Отже, кожного разу, коли в бік бібліотеки вносяться нові зміни, Клієнт повинен буде внести деякі відповідні зміни в кінці і перекомпілювати код. Звучить погано? Це дуже погана практика дизайну.

Приклади фабричного методу.

1. Скажімо, в системі "Малювання", залежно від введення користувача, можна малювати різні зображення, такі як квадрат, прямокутник, коло. Тут ми можемо використовувати заводський метод для створення примірників залежно

від введення користувача. Для додавання нового типу фігури не потрібно змінювати код клієнта.

2. Ще один приклад: на сайті подорожей ми можемо забронювати квитки на поїзд, а також квитки на автобус та квиток на рейс. У цьому випадку користувач може вказати свій тип подорожі як "автобус", "поїзд" або "рейс". Тут ми маємо абстрактний клас ' AnyTravel ' зі статичною функцією члена ' GetObject ', який залежно від типу подорожі користувача, створить і поверне об’єкт ' BusTravel ' або ' TrainTravel '. " BusTravel " або " TrainTravel " мають загальні функції, такі як ім'я пасажира, походження, параметри призначення.

Шаблон обсервер.

Спершу розглянемо наступний сценарій, щоб зрозуміти модель спостерігачів.

Припустимо, що ми будуємо додаток крикет, який сповіщає глядачів про таку інформацію, як поточний рахунок, швидкість виконання і т.д. Припустимо, ми зробили два дисплея елементи CurrentScoreDisplay і AverageScoreDisplay. CricketData має всі дані (запуски, миски тощо), і кожного разу, коли дані змінюються, елементи відображення повідомляються про нові дані, і вони відображають останні дані відповідно.

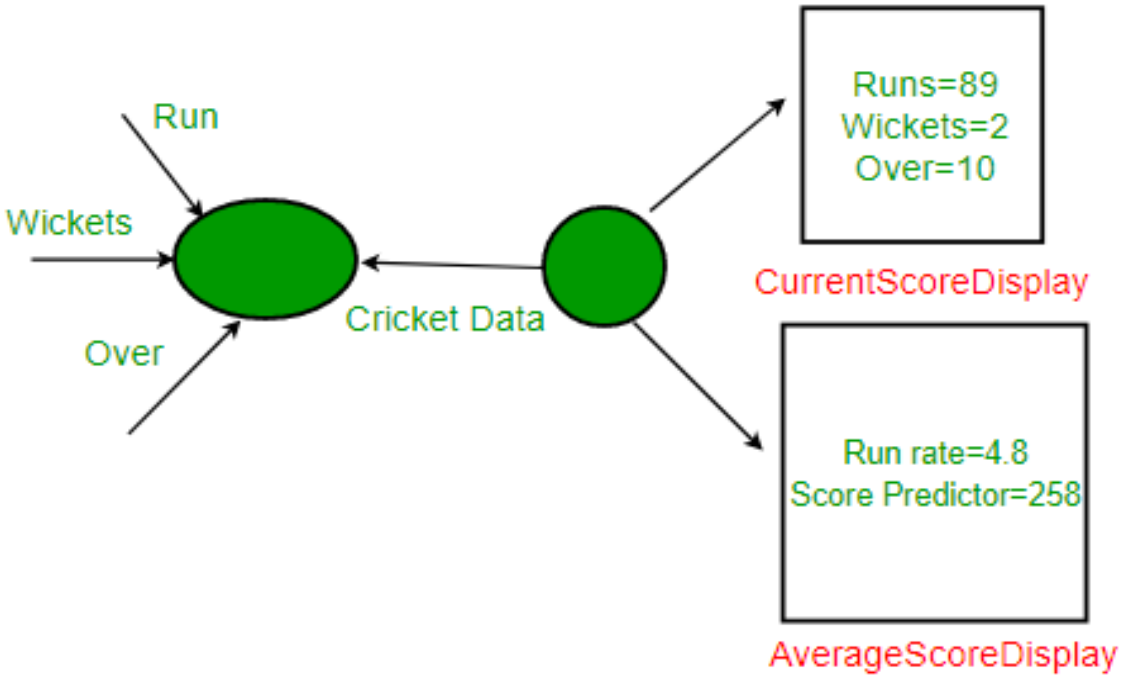


Рисунок 2.1 – Шаблон обсервер

Припустимо, що ми будуємо додаток крикет, який сповіщає глядачів про таку інформацію, як поточний рахунок, швидкість виконання і т.д. Припустимо, ми зробили два дисплея елементи CurrentScoreDisplay і AverageScoreDisplay. CricketData має всі дані (запуски, миски тощо), і кожного разу, коли дані змінюються, елементи відображення повідомляються про нові дані, і вони відображають останні дані відповідно

Застосування шаблону спостерігача до вищезазначеної проблеми: Подивимось, як ми можемо покращити дизайн нашої програми за допомогою шаблону обсервер. Якщо ми спостерігаємо потік даних, ми можемо легко побачити, що елементи CricketData та відображення дотримуються взаємозв'язку суб'єкт-спостерігач.

Шаблон декоратора.

Щоб зрозуміти патерн декораторів, давайте розглянемо сценарій, натхненний книгою "Головний шаблон проектування". Припустимо, ми створюємо додаток для магазину піци, і нам потрібно моделювати їхні класи піци. Припустимо, вони пропонують чотири типи піц, а саме: Peppy Paneer,

Farmhouse, Margherita та Fiesta з куркою. Спочатку ми просто використовуємо успадкування та абстрагуємо загальну функціональність у класі Піца.

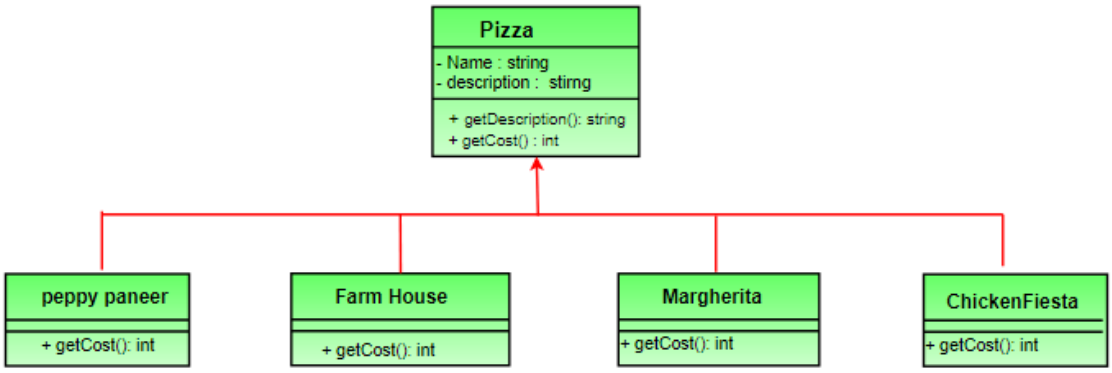


Рисунок 2.2 – Шаблон Декоратор

Кожна піца має різну вартість. Ми перекрили getCost ( ) у підкласах, щоб знайти відповідну вартість. Тепер припустимо нову вимогу, окрім піци, клієнт може також попросити кілька начинок, таких як свіжий помідор, барбекю тощо. Давайте задумаємося про те, що коли- небудь змінити у вищевказаних класах щоб клієнт міг вибрати піцу з начинками, і ми отримаємо загальну вартість піци та начинки, яку вибирає замовник.

Давайте розглянемо різні варіанти.

Варіант 1. Створіть новий підклас для кожного додавання піци. Діаграма класу виглядатиме так:

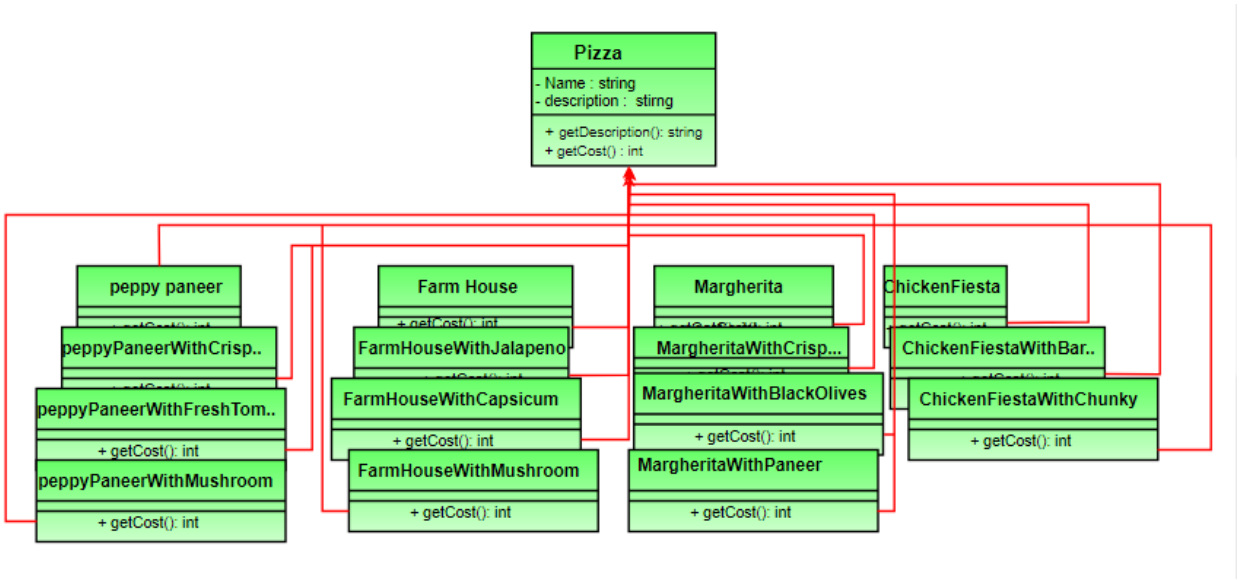


Рисунок 2.3 – Варіант використання шаблону декоратор

Це виглядає дуже складно. Занадто багато занять, і це кошмар з обслуговування. Крім того, якщо ми хочемо додати новий начинку або піцу, ми повинні додати стільки класів. Це, очевидно, дуже поганий дизайн.

Варіант 2: Додамо змінні екземпляри до базового класу піци, щоб представити, чи є кожна піца додатковою. Діаграма класу виглядатиме так:

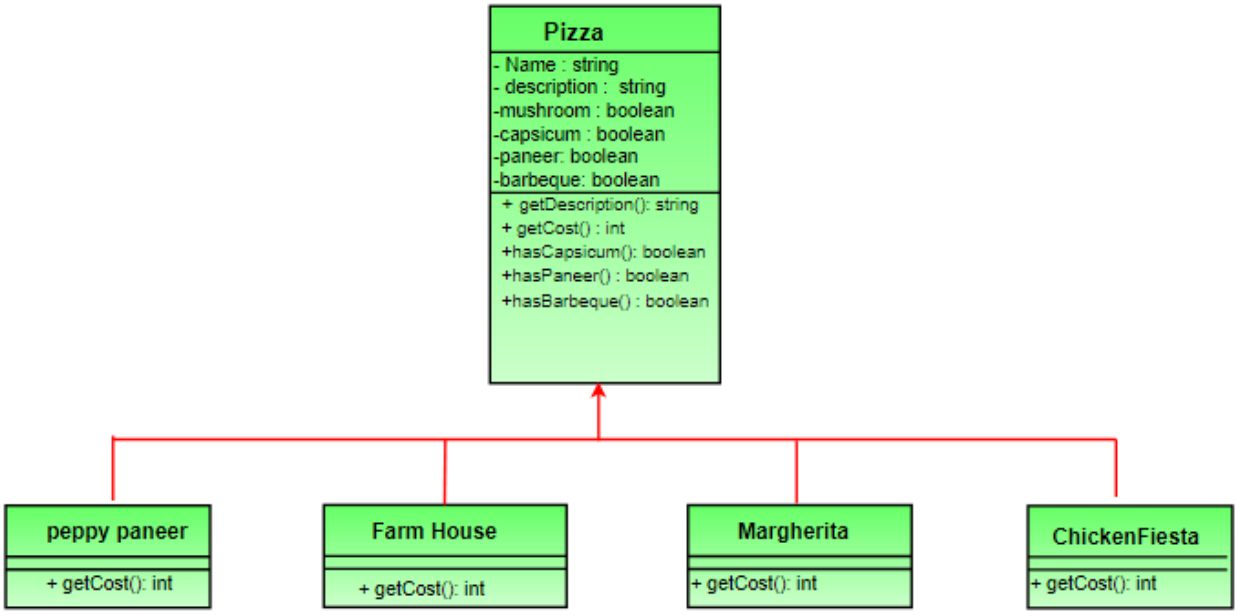


Рисунок 2.4 – Інший варіант використання шаблону Декоратор

GetCost ( ) з суперкласу обчислює витрати на всю начинку в той час як один в підкласі додає вартість цієї конкретної піци.

Отже, ми робимо піцу і «прикрашаємо» її начинками під час виконання:

1. Візьміть предмет піци.
2. «Прикрасьте» його об’єктом Capsicum.
3. «Прикрасьте» його об’єктом CheeseBurst.
4. Зателефонуйте getCost ( ) та використовуйте делегування замість спадкування для обчислення вартості начинок.

Зрештою, ми отримуємо піцу з начинками з чізбурсту та капсикуму. Візуалізуйте предмети «декоратора», як обгортки. Ось деякі властивості декораторів:

- Декоратори мають той же супер тип, що і предмет, який вони прикрашають.
- Ви можете використовувати кілька декораторів, щоб обернути предмет.
- Оскільки декоратори мають той самий тип, що й об’єкт, ми можемо пройти навколо прикрашеного об’єкту замість оригіналу.
- Ми можемо прикрашати предмети під час виконання.

Визначення:

Шаблон декоратора динамічно покладає додаткові обов'язки на предмет. Декоратори пропонують гнучку альтернативу підкласи для розширення функціональності.

Діаграма класу:

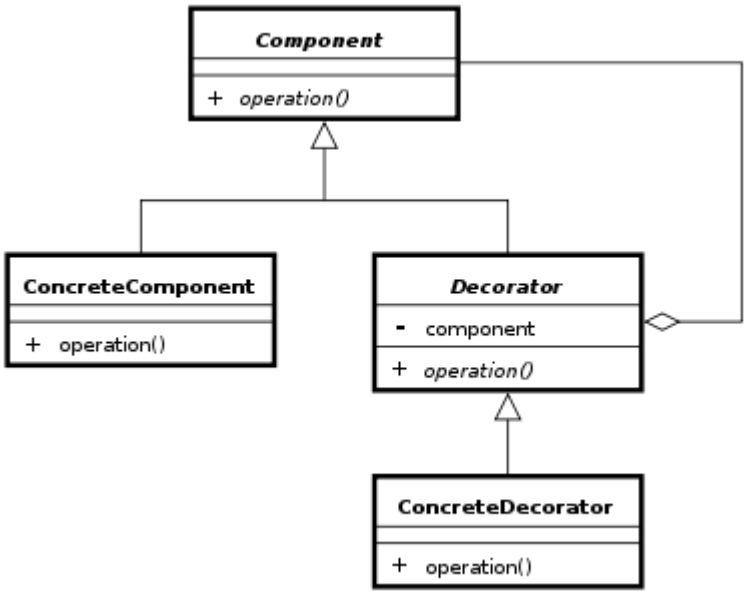


Рисунок 2.5 – Загальний вигляд шаблону Декоратор

- Кожен компонент може використовуватися самостійно або може бути обгорнутий декоратором.
- Кожен декоратор має змінну екземпляра, яка містить посилання на компонент, який він прикрашає (відношення HAS-A).
- ConcreteComponent є об'єкт, який ми будемо динамічно прикрашати.

Переваги:

- Шаблон декораторів може бути використаний, щоб дати можливість розширити (прикрасити) функціональність певного об'єкта під час виконання.
- Шаблон декораторів є альтернативою підкласифікації. Підкласифікація додає поведінку під час компіляції, і зміна впливає на всі екземпляри оригінального класу; декорування може забезпечити нову поведінку під час виконання окремих об'єктів.
- «Декоратор» пропонує підхід до додавання обов'язків як платно. Замість того, щоб намагатися підтримувати всі передбачувані функції у складному, що настроюється класі, ви можете визначити простий клас та додавати функціонал поступово за допомогою об’єктів Decorator.

Недоліки:

- Декоратори можуть ускладнити процес інстанції компонента, тому що вам не тільки доведеться інстанціювати компонент, але і загорнути його в ряд декораторів.
- Це може бути складно, коли декоратори слідкують за іншими декораторами, оскільки оглянути назад у кілька шарів ланцюжка декораторів починає виштовхувати візерунок декораторів за межі його справжнього наміру.

Шаблон Стратегія.

Як завжди, ми дізнаємось цю закономірність, визначаючи проблему та використовуючи шаблон її вирішення. Припустимо, ми будуємо гру «Street Fighter». Для простоти припустимо, що персонаж може мати чотири ходи, це удар, удар, перекидання та стрибок. У кожного персонажа є рухи ударами та ударами, але перекидання та стрибки необов'язкові. Як би ви моделювали свої заняття? Припустимо, спочатку ви користуєтеся успадкуванням і абстрагуєте загальні риси класу Fighter та дозволяєте іншим персонажам підклас класу Fighter.

Клас винищувача буде мати за замовчуванням реалізацію звичайних дій. Будь-який персонаж із спеціалізованим рухом може змінити цю дію у своєму підкласі. Діаграма класів була б така:

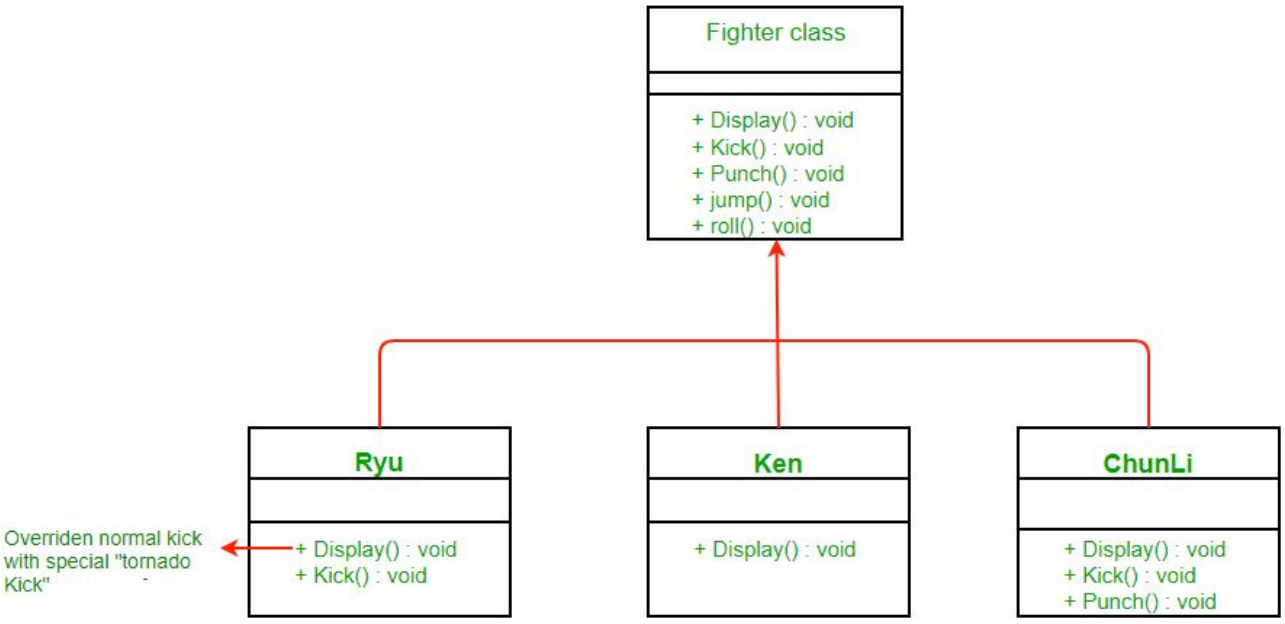


Рисунок 2.6 – Вигляд шаблону Стратегія

Які проблеми з вищезазначеним дизайном?

Що робити, якщо персонаж не виконує стрибок? Він все ще успадковує стрибкову поведінку від суперкласу. Хоча ви можете перекрити стрибок, щоб нічого не робити в цьому випадку, але, можливо, вам доведеться зробити це для багатьох існуючих класів і подбати про це і для майбутніх класів. Це також ускладнить обслуговування. Тому ми не можемо використовувати наслідування тут.

Що з інтерфейсом?

Погляньте на наступний дизайн:

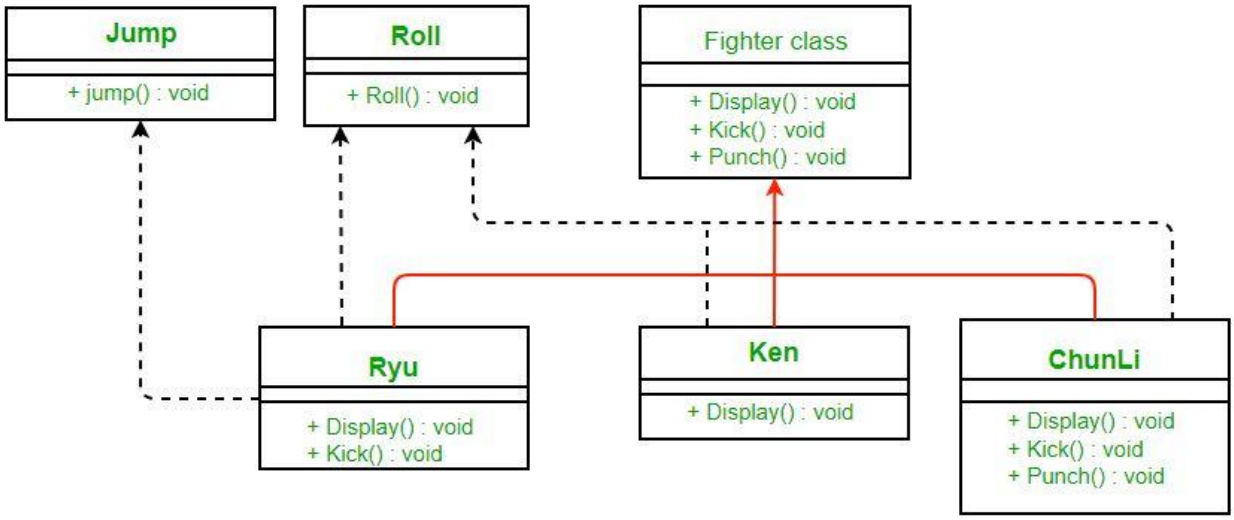


Рисунок 2.7 – Покращена структура патерну Стратегія

Це набагато чистіше. Ми вивели деякі дії (які деякі персонажі можуть не виконувати) з класу Fighter і зробили для них інтерфейси. Таким чином, лише символи, які повинні стрибати, реалізують JumpBehavior.

Які проблеми з вищезазначеним дизайном?

Основна проблема вищезазначеної конструкції - повторне використання коду. Оскільки не існує реалізація не по замовчуванням стрибати і рулонне поведінку ми можемо мати код лукавство. Можливо, вам доведеться переписувати однакове поведінку стрибків знову і знову у багатьох підкласах.

Як ми можемо цього уникнути?

Що робити, якщо ми створили класи JumpBehavior та RollBehavior замість інтерфейсу? Ну, тоді нам доведеться використовувати багатократне успадкування, яке не підтримується багатьма мовами через багато проблем, пов'язаних з ним.

Тут нам допомагає модель стратегії. Ми дізнаємося, що таке стратегія, а потім застосуємо її для вирішення нашої проблеми.

Визначення:

Вікіпедія визначає шаблон стратегії як:

"У комп'ютерному програмуванні модель стратегії (також відома як модель політики) - це модель дизайну програмного забезпечення, яка дозволяє вибирати поведінку алгоритму під час виконання. Шаблон стратегія:

- визначає сімейство алгоритмів,
- інкапсулює кожен алгоритм та
- робить алгоритми взаємозамінними в межах цієї родини ».

Діаграма класу:

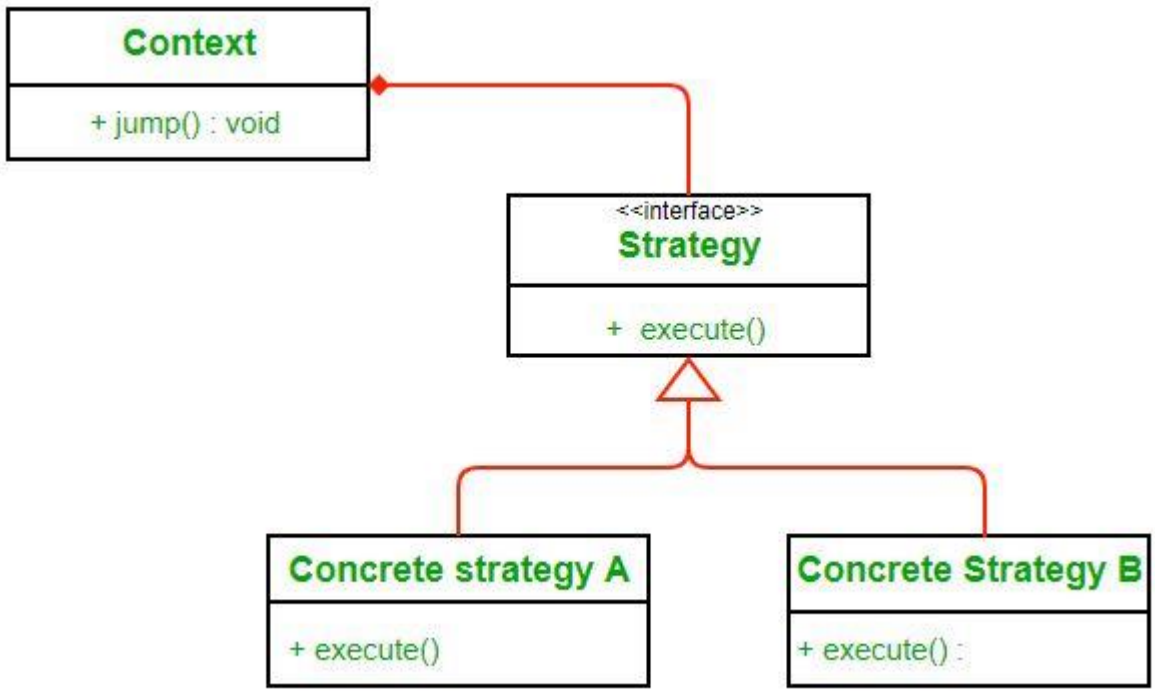


Рисунок 2.8 – Стандартна діаграма класів шаблону стратегія

Тут ми покладаємось на композицію, а не на наслідування для повторного використання. Контекст складається із Стратегії. Замість того, щоб реалізувати поведінку, Контекст делегує її Стратегії. Контекстом був би клас, який вимагав би змінити поведінку. Ми можемо динамічно змінювати поведінку. Стратегія реалізується як інтерфейс, щоб ми могли змінити поведінку, не впливаючи на наш контекст.

Ми матимемо чітке розуміння структури стратегії, коли будемо використовувати її для вирішення нашої проблеми.

Переваги:

1. Сімейство алгоритмів можна визначити як ієрархію класів і їх можна використовувати взаємозамінно для зміни поведінки програми без зміни його архітектури.

2. Інкапсулювавши алгоритм окремо, нові алгоритми, що відповідають одному інтерфейсу, можна легко ввести.

3. Додаток може перемикати стратегії під час виконання.

4. Стратегія дає змогу клієнтам вибрати необхідний алгоритм, не використовуючи оператор “switch” або ряд операторів “if-else”.

5. Структури даних, що використовуються для реалізації алгоритму, повністю укладені в класи Стратегії. Тому реалізацію алгоритму можна змінити, не впливаючи на клас контексту.

Недоліки:

1. Додаток повинен знати про всі стратегії, щоб вибрати правильну для правильної ситуації.

2. Контекст і класи стратегії зазвичай спілкуються через інтерфейс, визначений абстрактним базовим класом стратегії. Базовий клас стратегії повинен розкривати інтерфейс для всіх необхідних форм поведінки, які деякі конкретні стратегічні класи можуть не реалізувати.

3. У більшості випадків додаток налаштовує Контекст з необхідним об'єктом стратегії. Тому додатку потрібно створити та підтримувати два об’єкти замість одного.

Перший крок - визначити поведінку, яка може залежати від різних класів у майбутньому та відокремити їх від решти. Для нашого прикладу, нехай вони поводяться на ноги та стрибки. Щоб розділити ці поведінки, ми виведемо обидва методи із класу винищувачів та створимо новий набір класів, який представлятиме кожну поведінку (рис. 2.9).

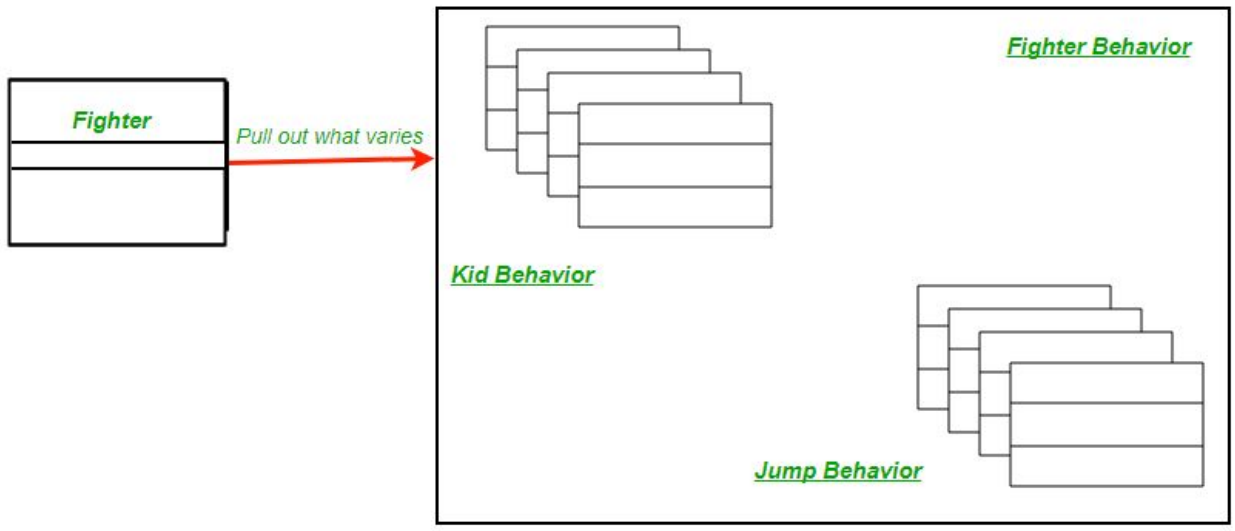


Рисунок 2.9 – Патерн Стратегія в прикладній задачі

Клас Fighter тепер делегує свою поведінку відштовхуванням і стрибками замість використання методів удару та стрибка, визначених у класі Fighter або його підкласі.

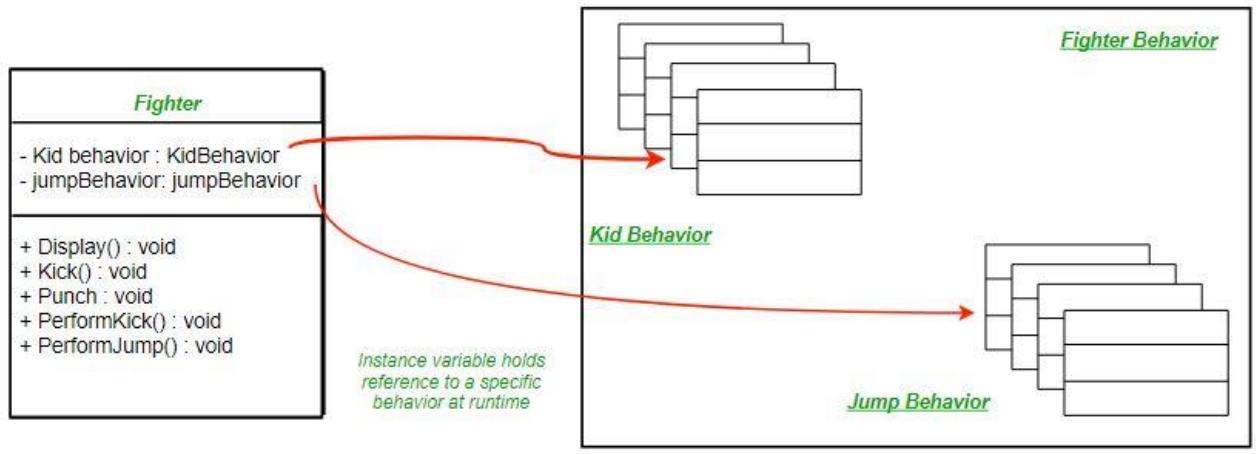
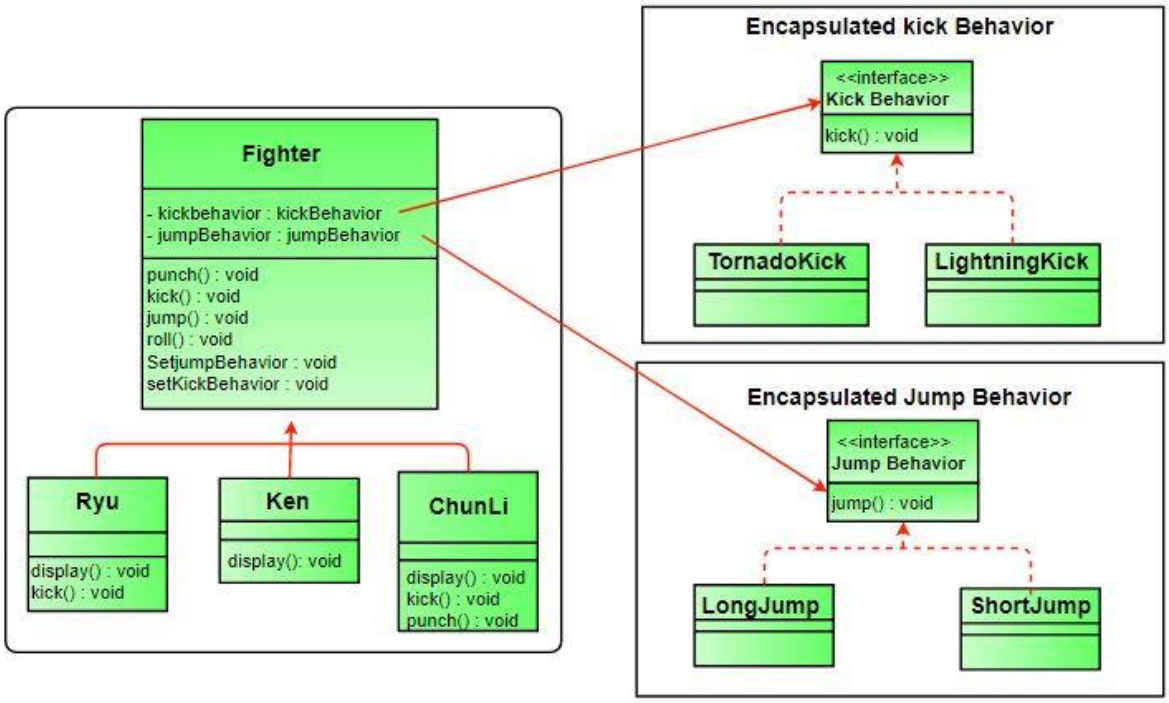


Рисунок 2.10 – Інтерфейс та реалізація шаблону Стратегія

Після переробки остаточна діаграма класу буде, як на рис. 2.11.



Риснок 2.11 – Фінальний вигляд патерну Стратегія

Порівнюючи нашу конструкцію з визначенням моделі стратегії інкапсульованої поведінки удару та стрибка - це два сімейства алгоритмів. І ці алгоритми взаємозамінні, як це очевидно в реалізації.

Шаблон адаптера.

Цю закономірність легко зрозуміти, оскільки реальний світ переповнений адаптерами. Для прикладу розглянемо USB до адаптера Ethernet. Це нам потрібно, коли у нас є інтерфейс Ethernet на одному кінці та USB на іншому. Оскільки вони несумісні між собою. ми використовуємо адаптер, який перетворює один на інший. Цей приклад є досить аналогічним об'єктово-орієнтованим адаптерам. У дизайні адаптери використовуються, коли у нас є клас (клієнт), який очікує певного типу об'єкта, і у нас є об'єкт (Adaptee), який пропонує ті самі функції, але піддається іншому інтерфейсу.

Для використання адаптера:

1. Клієнт робить запит до адаптера, викликаючи метод на ньому за допомогою цільового інтерфейсу.

2. Адаптер перекладає цей запит на адаптера за допомогою інтерфейсу адаптера.

3. Клієнт отримує результати дзвінка і не знає про наявність адаптера.

Визначення:

Шаблон адаптера перетворює інтерфейс класу в інший інтерфейс, який очікують клієнти. Адаптер дозволяє класам працювати разом, що не могло інакше через несумісні інтерфейси.

Діаграма класу:

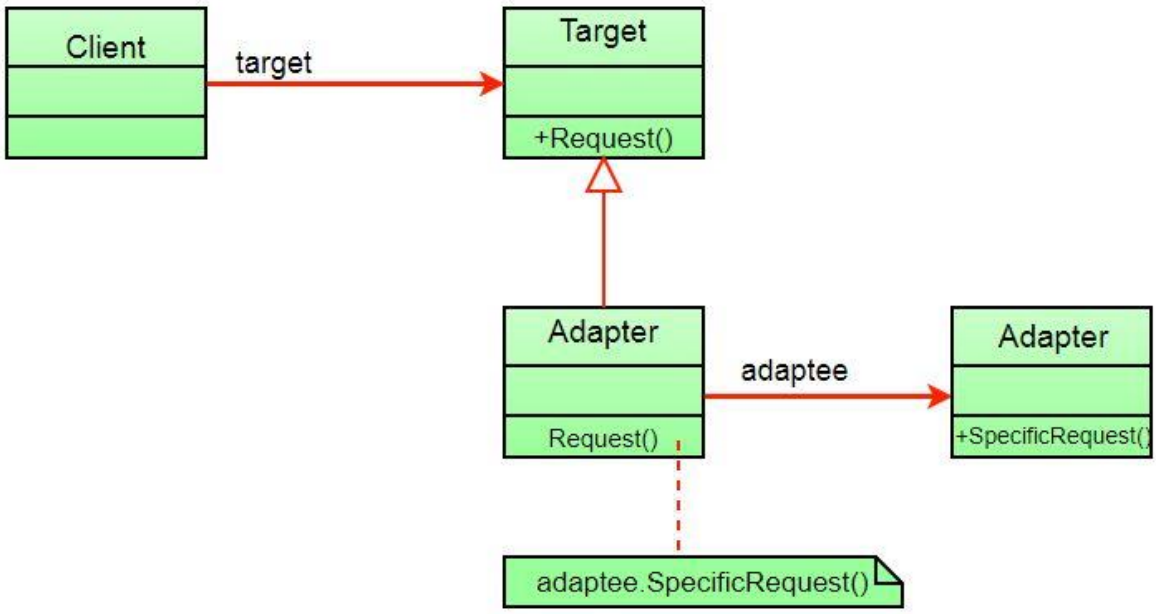


Рисунок 2.12 – Діаграма класів для патерну Адаптер

Клієнт бачить лише цільовий інтерфейс, а не адаптер. Адаптер реалізує цільовий інтерфейс. Адаптер делегує всі запити на Adaptee.

Шаблон проектування проксі.

Проксі означає "замість", що представляє "або" замість "або" від імені "- це буквальне значення проксі, що безпосередньо пояснює модель дизайну проксі. Проксі також називають сурогатами, ручками та обгортками. Вони за

структурою тісно пов'язані з адаптерами та декораторами, але не за призначенням.

Прикладом реального світу може бути чек або кредитна картка - проксі для того, що є на нашому банківському рахунку. Він може використовуватися замість готівки та надає засіб доступу до цієї готівки, коли це вимагається. І саме це робить шаблон проксі - " Керує доступом до об'єкта, який вони захищають ".

Поведінка.

Як і в оздоблювальній схемі, проксі можуть бути пов'язані між собою. Клієнт і кожен проксі вважають, що це делегування повідомлень на реальний сервер.

Шаблон проксі-сервера використовується, коли нам потрібно створити обгортку для покриття складності основного об'єкта від клієнта.

Види проксі.

Віддалений проксі. Вони несуть відповідальність за віддалене представлення об'єкта, розташованого на відстані. Якщо говорити з реальним об'єктом, це може включати в себе маршалінг та демонтаж даних та розмову з віддаленим об'єктом. Вся ця логіка інкапсульована в цих проксі, і клієнтська програма не повинна про них турбуватися.

Віртуальний проксі. Ці проксі надають деякі типові та миттеві результати, якщо реальному об'єкту потрібно певний час для отримання результатів. Ці проксі-сервери ініціюють операцію над реальними об'єктами та надають додатку результат за замовчуванням. Після того як реальний об’єкт зроблений, ці проксі-сервери передають фактичні дані клієнту, де він надав фіктивні дані раніше.

Проксі-захист:

Якщо програма не має доступу до якогось ресурсу, такі проксі-сервери спілкуватимуться з об'єктами в додатках, які мають доступ до цього ресурсу, а потім отримують результат.

Smart Proxy. Розумний проксі забезпечує додатковий рівень безпеки, розміщуючи конкретні дії під час доступу до об'єкта. Прикладом може бути

перевірка того, чи заблокований реальний об'єкт до його доступу, щоб переконатися, що жоден інший об'єкт не може його змінити.

РОЗДІЛ 3
РЕАЛІЗАЦІЯ РЕПОЗИТОРІЮ ПАТЕРНІВ ПРОГРАМНИХ АРХІТЕКТУР

3.1 Концепція побудови репозиторію

Наша задача являє собою побудову програмного комплексу репозитарію патернів і архітектур. Для початку визначимося що така репозитарій.

Репозиторій(сховище) - місце, де зберігаються і підтримуються будь-які дані. Найчастіше дані в репозиторії зберігаються у вигляді файлів, доступних для подальшого поширення по мережі.

Отже у нас буде репозиторій (сховище) архітектур і патернів які можна використати в структурі архітектур. А наш програмний комплекс дозволить створювати і керувати репозитарієм. Створювати редагувати і видаляти, архітектури і патерни.

Побудуєм структурну організацію програмного комплексу створення репозиторію (рисунок 3.1).

Тепер можна переходити до створення програмного забезпечення.

3.2 Опис функціональної структури системи

В системі “Архітектор ПС” наявні три типи користувачів:

– адміністратор – додає у базу нові архітектури, шари, компоненті та патерни та обслуговує базу даних архітектур і систему в цілому.

– архітектор – комбінує патерни, для вирішення класів специфікованих задач шляхом формування наборів альтернативних архітектур.

– експерт – оцінює сформовані архітектором множини альтернативних архітектур програмних додатків за певними визначеними критеріями якості.

“Архітектор програмних систем” складається з трьох підсистем:

1. Підсистема управління репозиторієм патернів (адміністратор).
2. Підсистема створення альтернативних архітектур (архітектор).
3. Підсистема оцінювання архітектур (експерт).

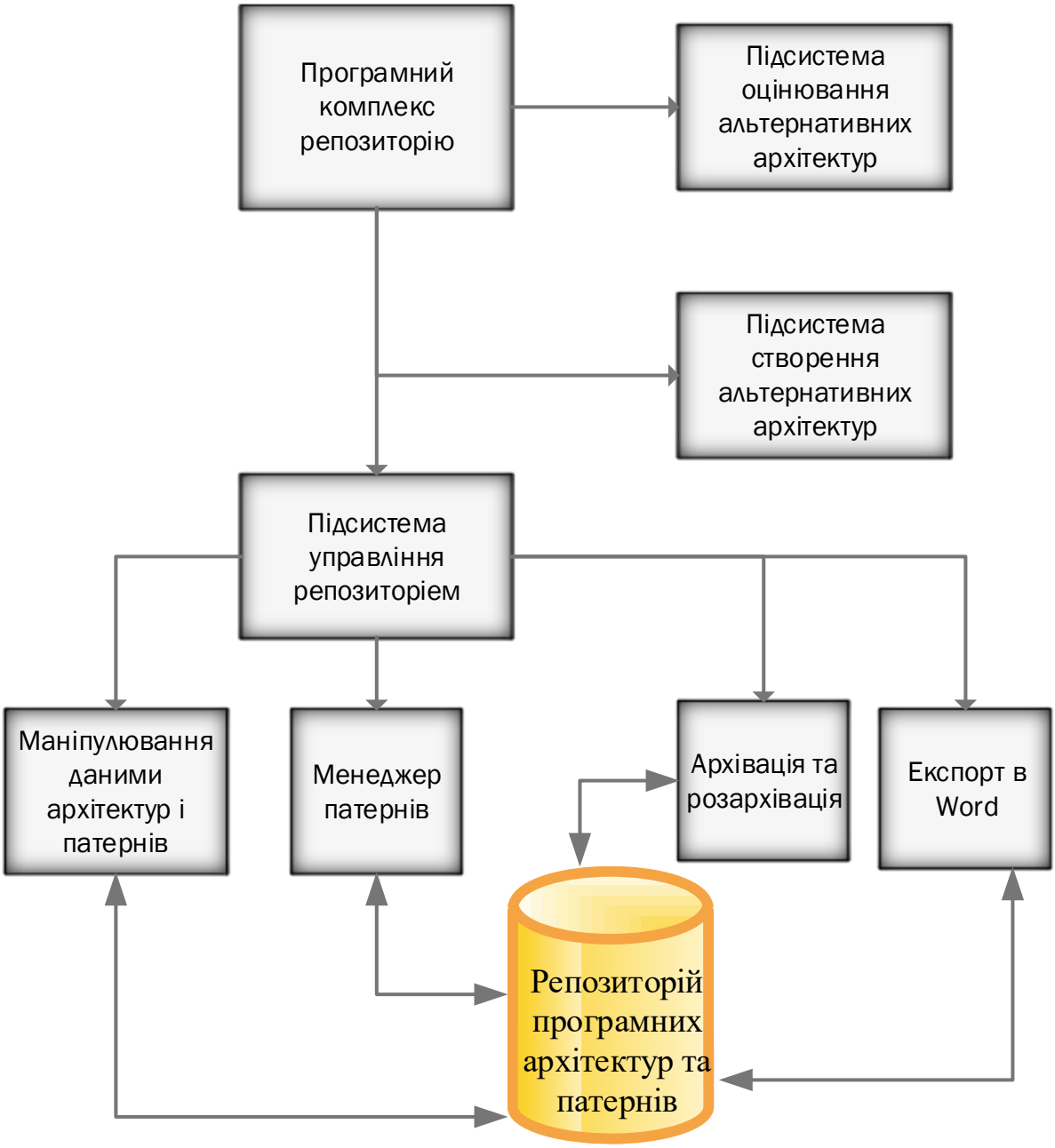


Рисунок 3.1 – Структура репозиторію шаблонів

Створимо структуру бази даних репозиторію (рисунок 3.2).

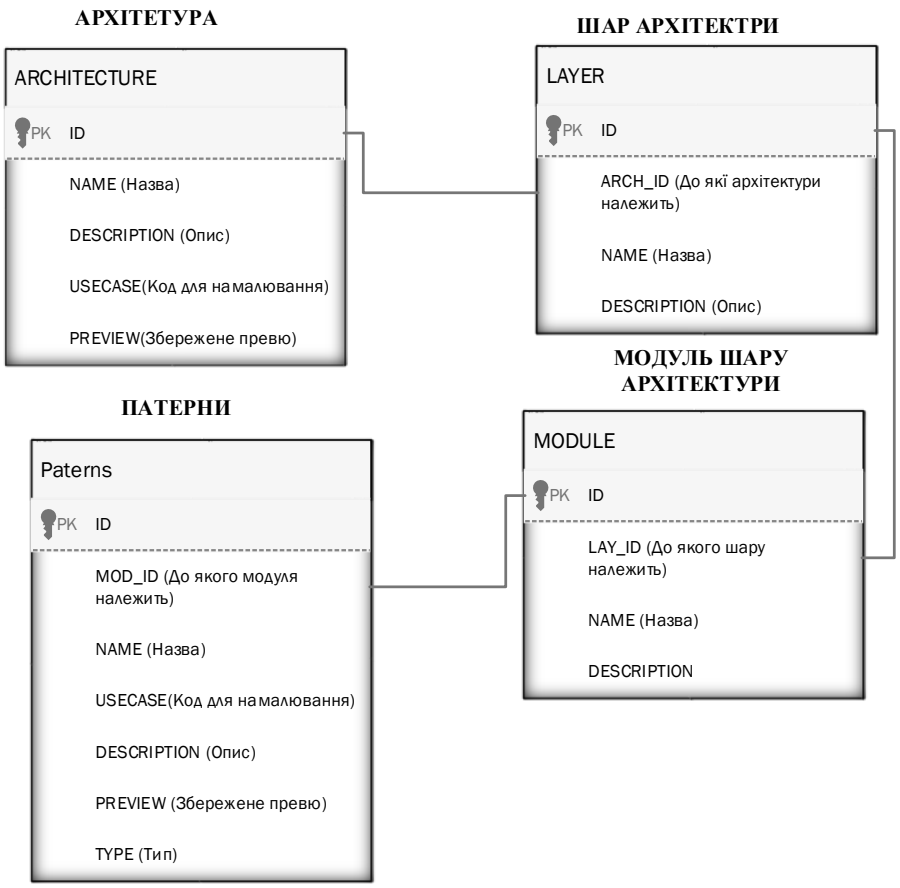


Рисунок 3.2 – Структура БД репозиторію патернів

Для проєктування і подальшої побудови об’єктно-орієнтованої системи застосовано графічне моделювання з використанням діаграм UML. При розробці статичної моделі об’єктів системи використовувались діаграми класів, а для розробки функціональних вимог – use case діаграми (діаграми сценаріїв).

функціональні можливості адміністратора репозиторію патернів забезпечуються використанням підсистеми управління репозиторієм патернів. Діаграма варіантів використання роботи адміністратора показана на Рисунок 3.1

Актор:

– Адміністратор – власне адміністратор репозиторію патернів який займається роботою з базою даних (БД) і керує БД архітектур.

– Сценарії Use case:

– Редактор архітектур – базова функція редагування репозиторію патернів.

– Створення БД – створення нової БД архітектур патернів. Спочатку адміністратор обирає, де створити БД, після чого підсистема створює БД.

– Під’єднання до БД – підключення до існуючої БД архітектур.

– Від’єднання від БД – відключення від БД, коли підключення встановлено.

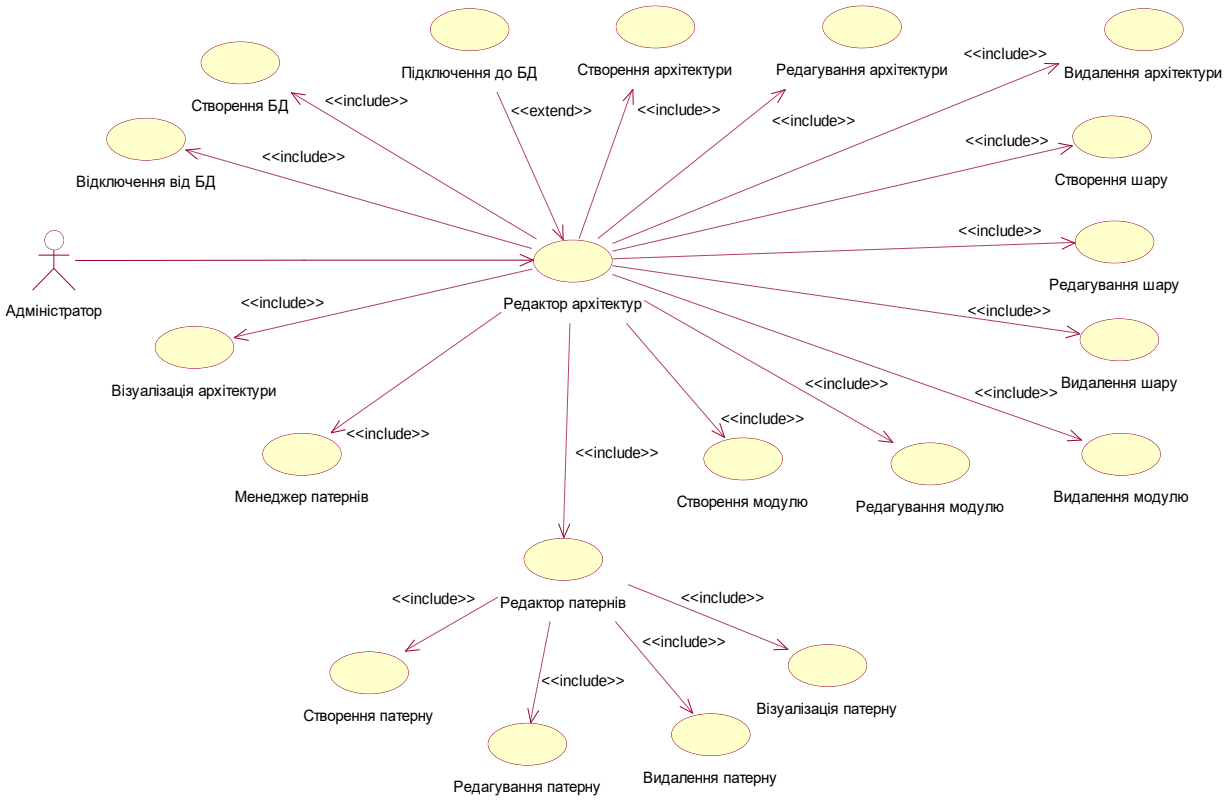


Рисунок 3.1 – Діаграма використання системи роллю Адміністратор

– Створення архітектури – створення нової архітектури в БД.

– Редагування архітектури – редагувати існуючу архітектуру в БД.

– Видалення архітектури – видалення існуючої архітектури, що вибрана зі списку архітектур.

– Збереження архітектури – збереження внесених змін у архітектуру додатку.

– Створення шару – створення шару в архітектурі, обраній для редагування.

– Редагування шару – редагування існуючого шару архітектури додатку.

– Видалення шару – видалення шару з архітектури, відкритої для редагування.

– Створення компонентів – створення компонентів (категорії) у відкритій архітектурі.

– Редагування компонентів – редагування компонентів у відкритій архітектурі.

– Видалення компонентів – видалення компонентів з відкритої архітектури.

– Менеджер патернів – менеджер патернів для зручного управління патернами в компонентах (категоріях) архітектур програмних додатків.

– Візуалізація архітектури – відображення вигляду поточної архітектури, що редагується.

– Редактор патернів – редактор патернів викликається при редагуванні набору патернів поточного компонентів (категорії):

- Створення патерну – створення нового патерну.
- Редагування патерну – редагування вже існуючого патерну.
- Видалення патерну – видалення патерну вибраного зі списку.
- Візуалізація патерну – візуалізувати текстовий опис у структуру патерну.
- Збереження патерну – зберегти внесені зміни до патерну в БД.

3.3 Обгрунтування інструментів розробки

В якості інструментальної платформи для розробки “Архітектора програмних систем” була обрана Інтегроване середовище розробки IDE Eclipse призначене спеціально для розробки на мові Java, перераховується як одна з найбільших трьох Java IDE. Сучасне інтегроване середовище розробки доступне як для настільних, так і хмарних видань. Хмарне видання Eclipse під назвою Eclipse Che дозволяє програмістам розробляти програми через веб-браузер.

Обидва випуски Eclipse IDE оснащуються необхідною / додатковою функціональністю з використанням плагінів. На ринку Eclipse існує маса

плагінів, доступних для IDE. Для полегшення поступової компіляції коду Java Eclipse оснащений спеціальним компілятором.

Для Java-програмістів, які хочуть розробити конкретні функціональні можливості для Eclipse, доступний PDE (Plugin Development Environment). Щоб допомогти розробникам Java пришвидшити розробку додатків, Eclipse показує потужні інструменти для складання графіків, моделювання, звітування та тестування.

Eclipse підтримує розробку додатків для кількох мов програмування за допомогою плагінів. C, C ++, Groovy, Haskell, JavaScript, Perl, PHP, Ruby і Scala – деякі з різних мов програмування, що підтримуються Eclipse. Тому виберемо в якості розробки програми дане середовище.

UI розробки показано на Рисунок 3.2.

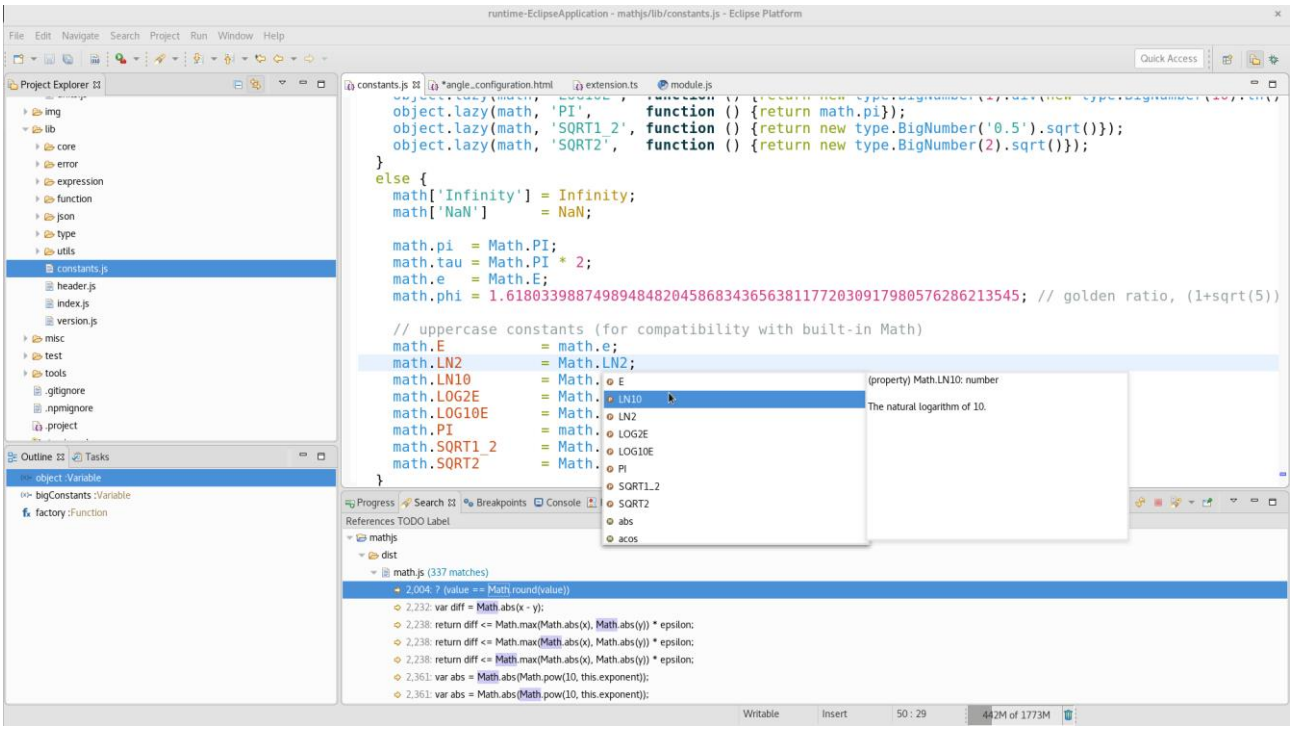


Рисунок 3.2 – Інтерфейс інтегрованого середовища розробки

3.4 Методика роботи системи

Після запуску “Архітектора програмних систем” з’являється головне вікно системи. Для початку роботи із системою користувач має обрати власну роль у системі і пройти етап авторизації.

Для кожного з користувачів доступні різні види операцій (функцій) всередині системи. Адміністратор має функції “Управління репозиторієм патернів” і “Створення альтернативнх архітектур”. Перша операція служить для редагування бази даних архітектур (репозиторію патернів) і забезпечується роботою підсистеми управління репозиторієм патернів. Друга операція існує для контролю та тестування доданого в репозиторій патернів контенту, а також контенту, що редагується. Ця операція забезпечується підсистемою створення альтернативних архітектур (Архітектор), яка буде розглянута у п.3.2.

Вибір функції “Управління репозиторієм патернів” показаний на рис. 3.3. В разі вибору цієї операції з’являється головне вікно підсистеми управління репозиторієм патернів, яке показано на Рисунок 3.3, де Адміністратору потрібно підключитися до БД архітектур, або створити нову БД.

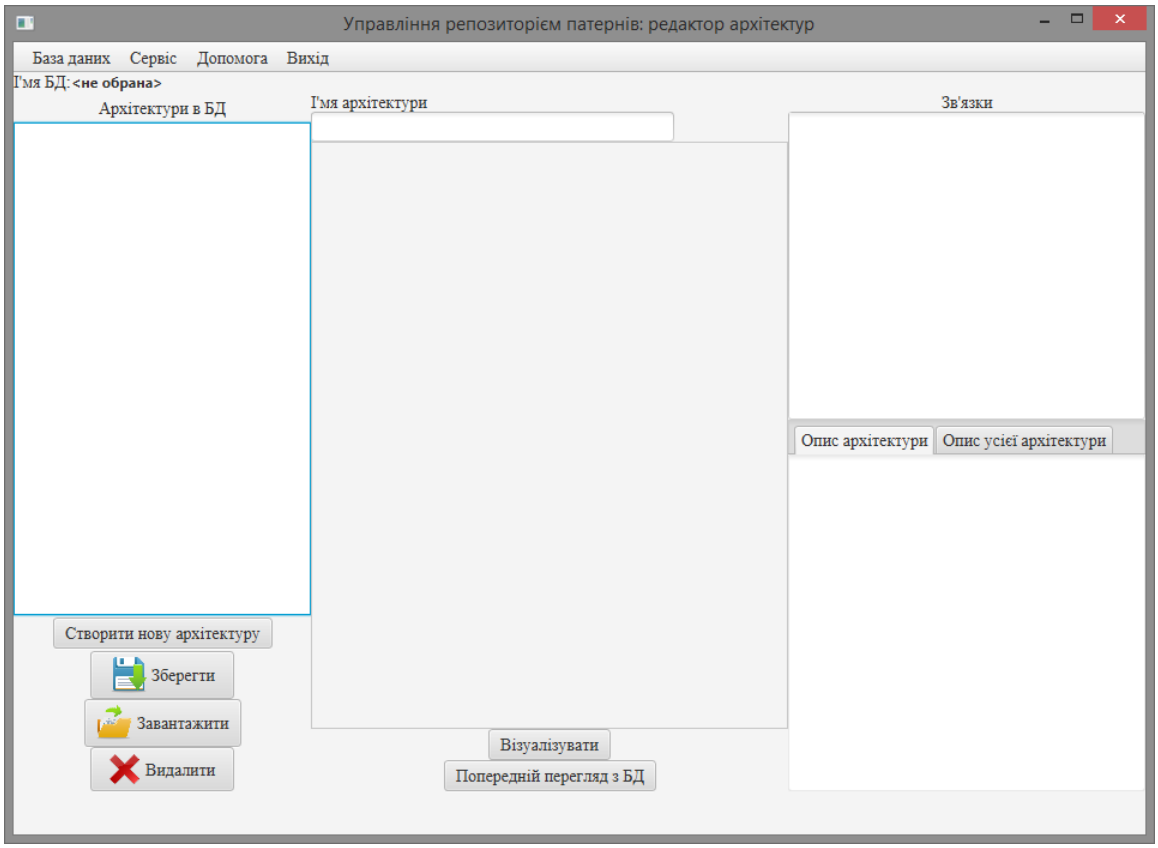


Рисунок 3.3 Головне вікно підсистеми управління репозиторієм патернів

Для створення нової БД у головному меню підсистеми слід обрати База даних і Створити нову БД.

Далі вибираємо папку в якій потрібно створити БД і, коли потрібно, створюємо. Натискаємо вибір папки, як показано на Рисунок 3.4.

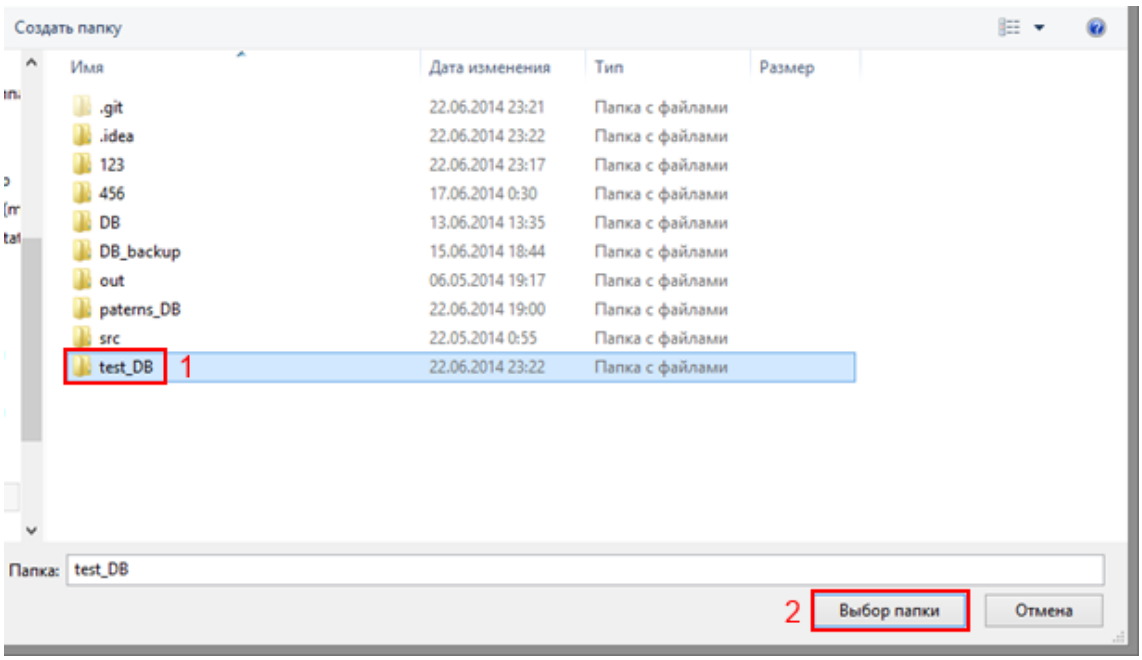


Рисунок 3.4 – Вибір папки (каталогу) для БД

Коли потрібно під'єднатися до БД, то в меню підсистеми обираємо База даних і Під'єднання до БД . Далі у вікні обираємо папку з БД, як показано на Рисунок 3.5.

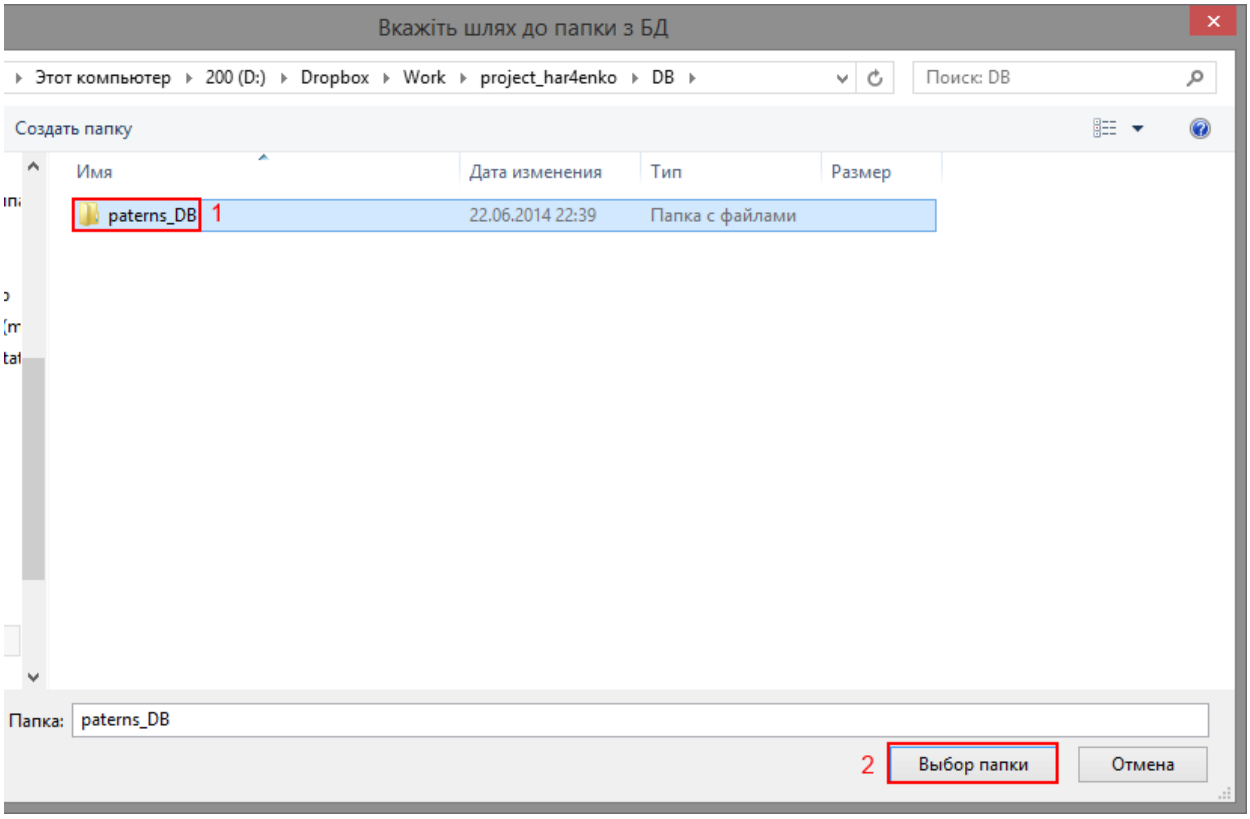


Рисунок 3.5 – Під’єднання до БД патернів архітектур

В разі, коли потрібно від'єднатися від БД, в меню слід вибрати База даних і Від'єднатися від БД. Після під'єднання до БД відкривається головне вікно редактора архітектури із структурою програмного додатка, що показано на Рисунок 3.6.

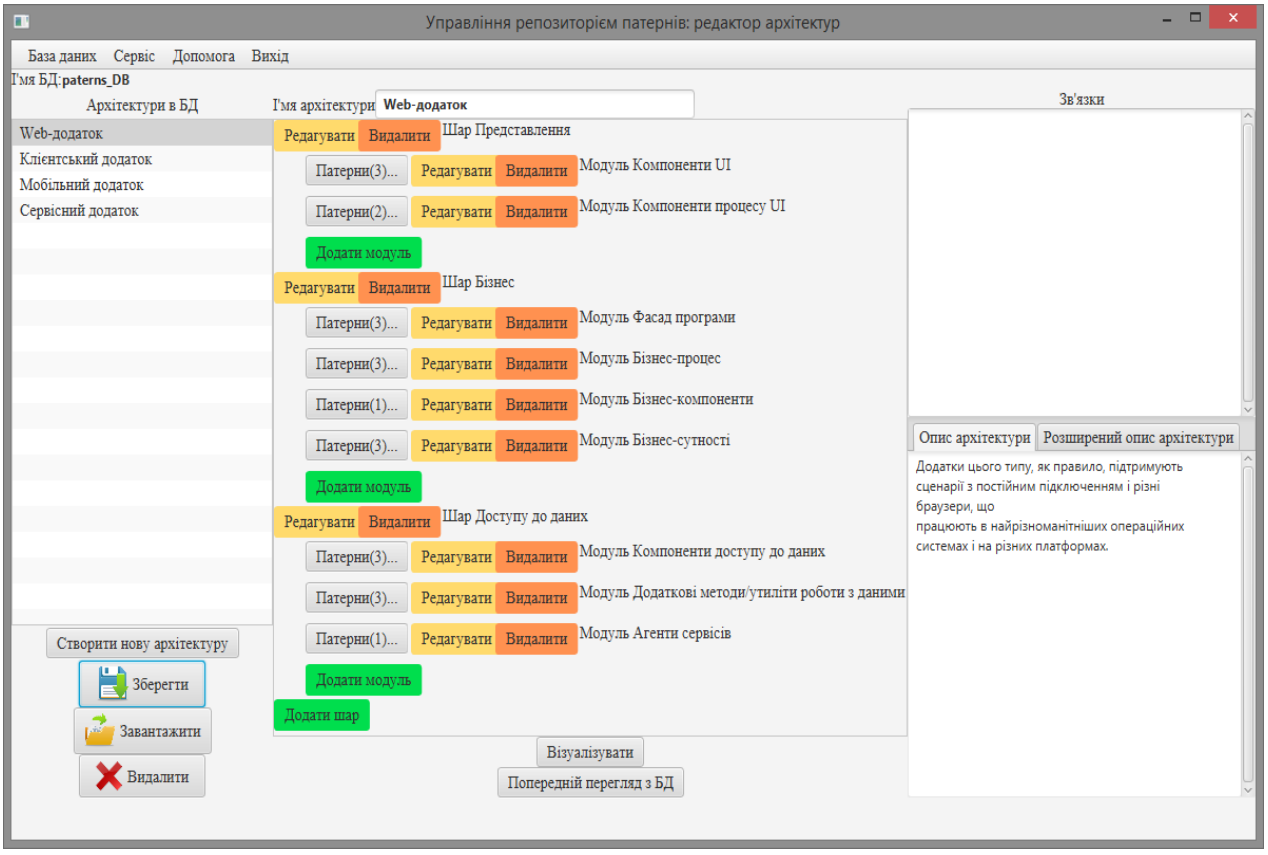


Рисунок 3.6 – Головне вікно редактора архітектури програмного додатка

Створення нової архітектури.

Для створення архітектури натисніть кнопку «Створити нову архітектуру», після чого слід дати їй ім’я і натиснути кнопку «ОК» у вікні «введення». Після цього нова архітектура додатку повинна з’явитися в списку «Архітектури в БД».

Редагування архітектури.

Натисніть на архітектуру і натисніть кнопку завантажити 

, або зробіть двійний клік лівою клавішею миші по імені архітектури в списку архітектур. Після цього побачимо архітектуру додатку у вікні його структури, де можна додавати шари чи компоненті, а також редагувати патерни компонентів, що видно із Рисунок 3.7.

Рисунок 3.7 – Вікно структури архітектури обраного програмного додатка

Основні елементи вікна редактору архітектур.

У вікні редактора архітектур крім вищеописаних елементів присутні, також, вікна для опису архітектури і створення зв'язків між шарами (або компонентами) додатку. Документація мови візуального представлення архітектур детально надана в пункті головного меню «Допомога» підсистеми управління репозиторієм патернів.

Під блоком структури архітектури знаходиться кнопка для перемальовування картинки архітектури. Також є кнопка для видалення виділеної архітектури 

і збереження внесених змін в архітектуру 

(див. рис.3.6).

Також в головному меню доступні додаткові сервісні утиліти:

«Оптимізація БД» видаляє всі шари, компоненті і патерни, в яких є проблеми зі зв'язками (патерн привязаний до неіснуючої архітектури тощо). «Менеджер патернів» слугує для управління патернами, а саме їх переміщення з однієї архітектури (компонентів) до іншої (іншого). «Експорт в Word 2007» експортує всі архітектури і патерни в файл Word 2007+ формату "docx" в

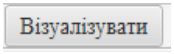
зручному для читання вигляді. Адміністратор обирає файл, куди зберігати дані, що експортуються.

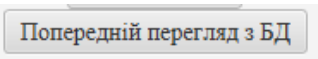
Опис деяких кнопок у вікні структури архітектури.

Додати новий компонент або шар: по кнопкам 

, 

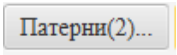
з'являється вікно, в якому потрібно ввести назву і натиснути введення, а в наступне вікно потрібно ввести опис шару або компонентів і натиснути введення. Після цього шар (або компонент) з'явиться в структурі архітектури програмного додатку.

Щоб візуалізувати внесені в архітектуру додатку зміни, натискаємо кнопку 

. Для того, щоб подивитись зображення, яке зараз збережене для поточної архітектури в БД, натискаємо 

. При натисканні кнопки 

підсистема перепитає, чи ви впевнені, що бажаєте видалити архітектуру.

Кнопка патерни 

запускає вікно роботи з патернами, де можна редагувати патерни обраного компонента. Разом з тим вікно редактора закривається, а всі зміни, внесені в архітектуру, зберігаються (бо компонент може бути новий). Після цього з'являється вікно редактора патернів, що показано на Рисунок 3.8. Робота у вікні редактора патернів.

Так виглядає вікно редагування патернів, де можна створювати, видаляти і редагувати патерни відповідного компонента (категорії) для шару нашого додатку.

Створення нового патерну.

Аби створити новий патерн, потрібно натиснути кнопку «Створити новий патерн», після чого ввести його ім'я та натиснути "ОК" у віконці "Введення". Після цього новий патерн для архітектури з'явиться у списку

Редагування існуючого патерну .

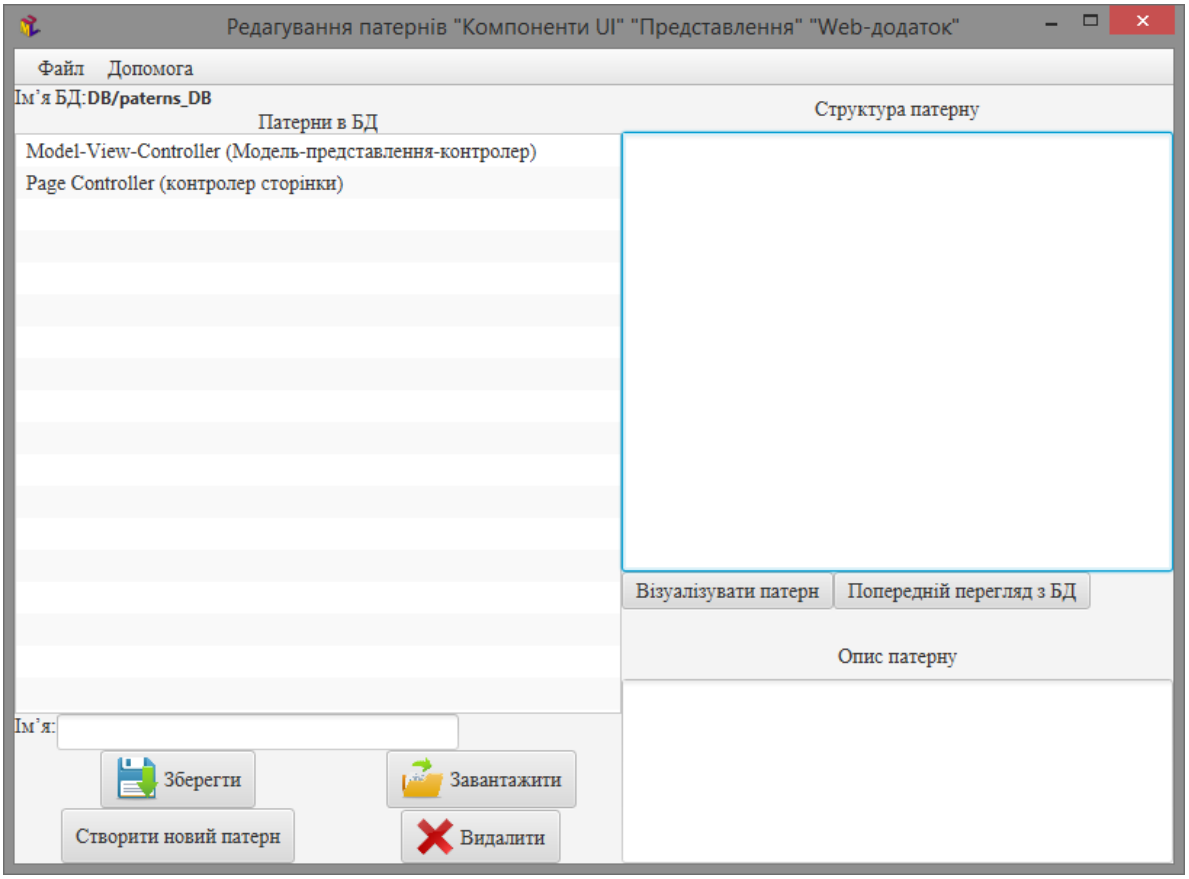


Рисунок 3.8 – Вікно редактора патернів

Для редагування патерну потрібно завантажити існуючий патерн. Для цього обираємо його в списку патернів і натискаємо кнопку завантаження, або робимо подвійний клік по імені патерну в списку лівою кнопкою миші. Тепер можна внести в патерн зміни, після чого натискаємо кнопку зберегти. Наявні такі кнопки:

– 

- Завантаження патерну.

– 

- Видалення патерну.

– 

- Збереження змін, внесених у патерн.

У формі вікна є поля для редагування структури та опису патерну. Патерн завжди можна завантажити шляхом подвійного натискання на його імені в списку. При виході з редактору патернів система запитає вас, чи ви впевнені, що хочете вийти. Разом з тим слід бути уважним, бо коли ви вийдете без попереднього збереження патерну, всі не збережені зміни буде втрачено.

Щоб візуалізувати внесені у структуру патерну зміни, натискаємо кнопку Візуалізувати патерн . Для того, щоб подивитись зображення, яке зараз збережене для патерну, натисніть Попередній перегляд з БД . Щоб закінчити редагування натисніть у меню поля Файл і Закрити .

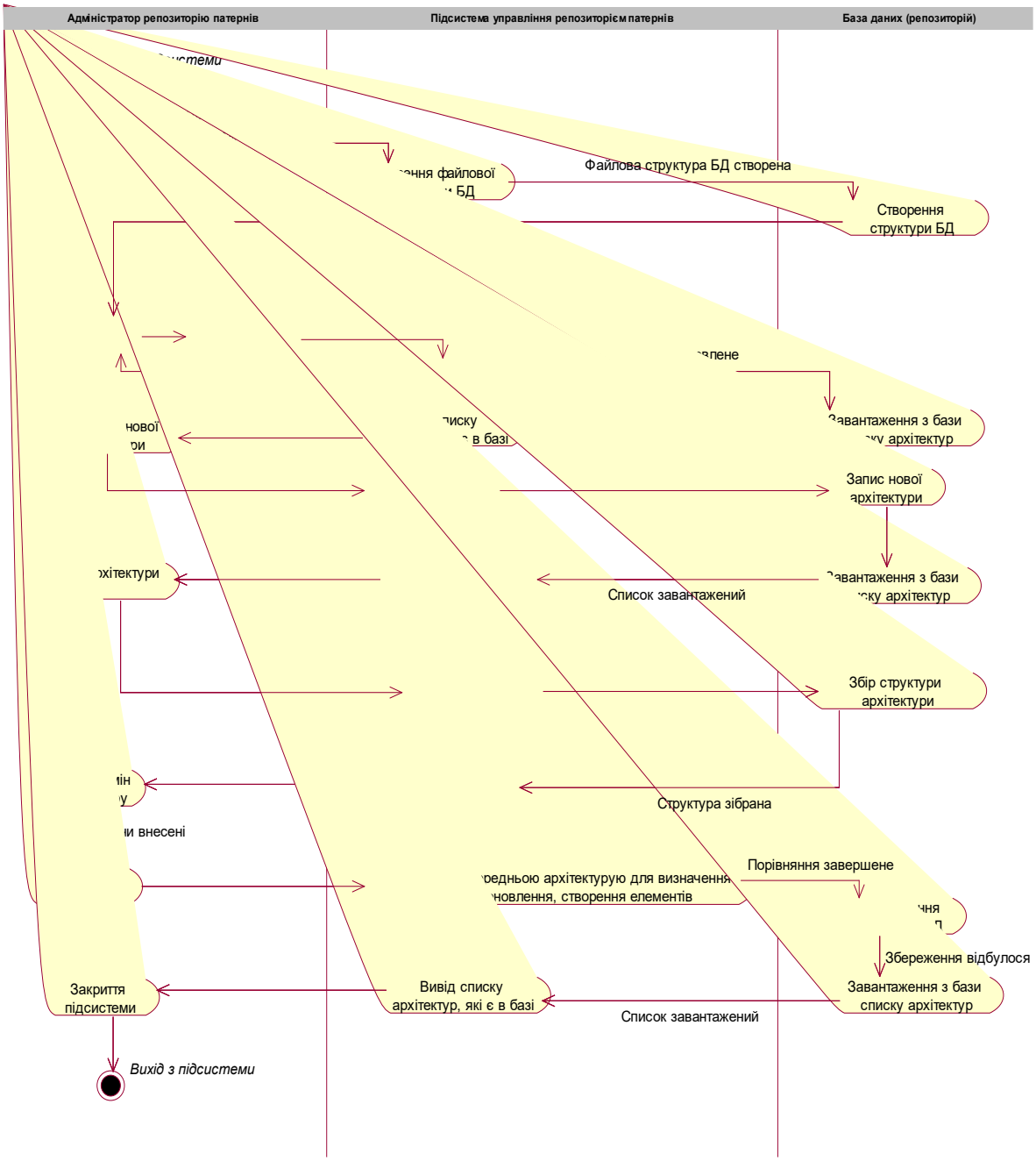


Рисунок 3.9– Діаграма активності Адміністратора

Опис діаграми активності «Сворення нової архітектури»

Функціональні можливості Адміністратора в процесі створення нової архітектури програмного додатку, або редагування існуючої, показані на UML-діаграмі активності на Рисунок 3.9, Рисунок 3.10.

Групи відповідальності на діаграмі активності:

– Адміністратор репозиторію патернів– дії, які виконує адміністратор.

– Підсистема управління репозиторієм патернів – дії, які виконуються ПС.

– База даних (репозиторій) – дії, які виконує БД .

– Дії (Activity):

– Вибір папки для БД – Адміністратор обирає папку для БД.

– Створення файлової структури БД – в папці створюється файлова структура бази даних репозиторію.

– Створення структури БД – викликається створення структури репозиторія в БД, а саме: створюються необхідні для роботи таблиці.

– Підключення до БД – підключитися до БД (вибір пункту меню: Підключення до БД).

– Вибір папки з БД – вибір папки з базою даних патернів архітектур.

– Завантаження з БД списку архітектур – обробка базою даних запиту на отримання списку архітектур.

– Вивід списку архітектур, які є в базі – вивід списку архітектур репозиторію.

– Створення нової архітектури – процес запиту у користувача імені і опису нової архітектури і подальше створення архітектури в базі даних.

– Запис нової архітектури в базу – формування запиту до БД на створення нової архітектури програмного додатку.

– Запис нової архітектури – обробка БД запиту на створення нової архітектури.

– Вибір архітектури на редагування – вибір архітектури на редагування (введення імені архітектури і натискання кнопки завантаження, або подвійний клік на імені архітектури; увага: Разом з тим не збережена попередня архітектура буде замінена).

– Запит структури архітектури з БД – створення запитів на збір інформації про архітектуру додатка.

– Збір структури архітектури – обробка запитів щодо збору інформації про архітектуру додатка.

– Відображення структури – відображення структури архітектури Адміністратору у відповідній зоні вікна підсистеми.

– Внесення змін в архітектуру – Адміністратор записує необхідні зміни в архітектуру програмного додатка.

– Збереження внесених змін – Адміністратор натискає кнопку збереження архітектури додатку.

– Порівняння з попередньою архітектурою для визначення видалення, оновлення, створення єлементів – підсистема порівнює колишній варіант архітектури з новим і, коли знаходить компоненті чи шари, що були видалені, видаляє їх з бази даних, а для внесених змін і добавлених компонентів формує запити на додавання їх в БД.

– Збереження в БД – обробка запитів на видалення і створення та збереження поточного стану БД архітектур.

– Закриття підсистеми управління репозиторієм патернів – користувач закриває підсистему роботи з репозиторієм.

Користуючись можливостями підсистеми управління репозиторієм патернів Адміністратор репозиторію патернів може виконувати дії по управлінню власне патернами, що показано на діаграмі активності, яка зображена на Рисунок 3.10.

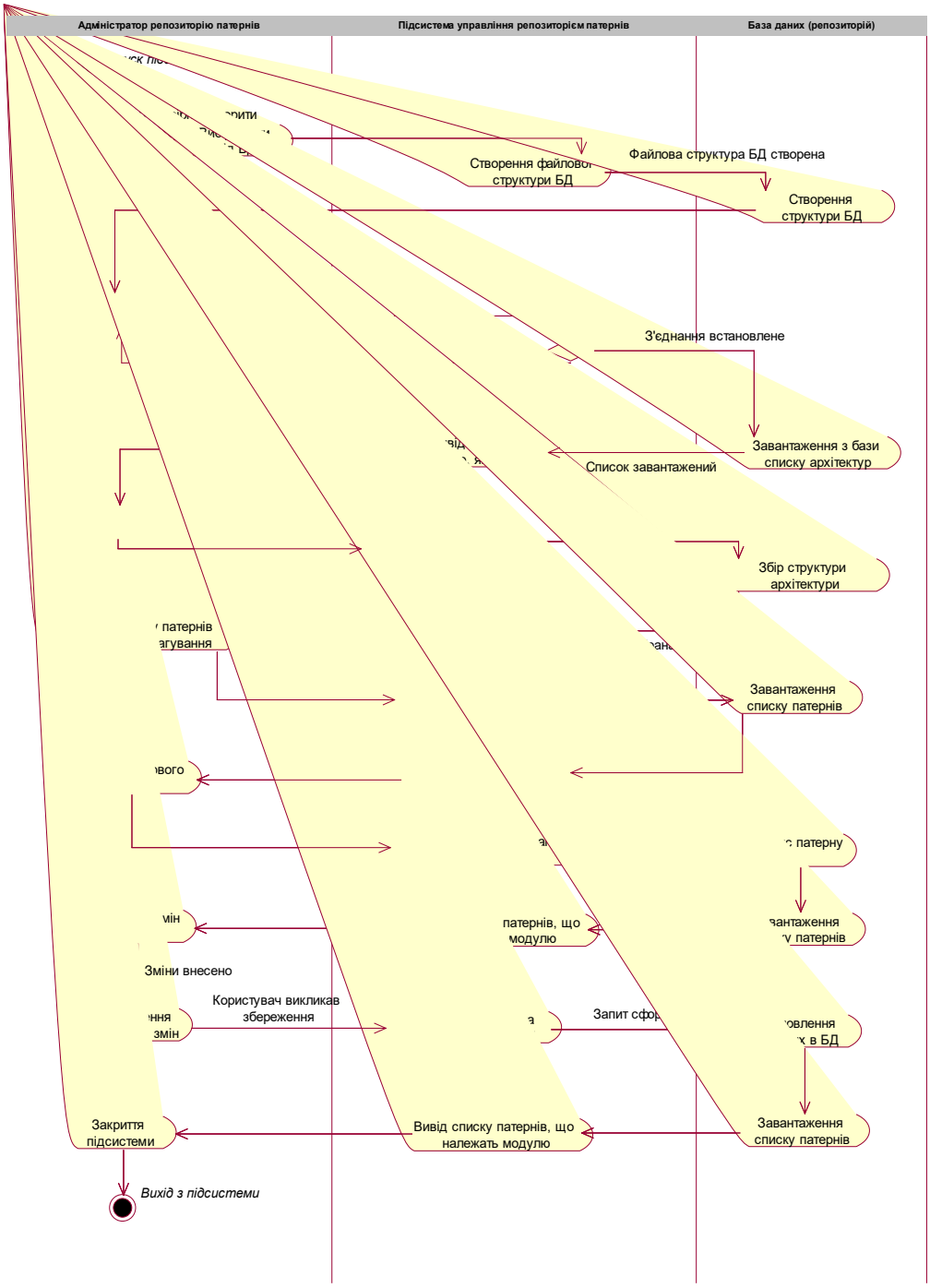


Рисунок 3.10 – Діаграма активності Адміністратора по управлінню репозиторієм

Опис діаграми активності «створення нового патерну»

Групи відповідальності на діаграмі активності:

- Адміністратор репозиторію патернів – дії, які виконує адміністратор.
- Підсистема управління репозиторієм патернів – дії, які виконує ПС.
- База даних (репозиторій) – дії, які виконує БД.
- Дії (Activity):

– Вибір папки під БД – адміністратор обирає папку під БД.

– Створення файлової структури – підсистема в папці створює файлову структуру БД.

– Створення структури БД – підсистема створює структуру БД репозиторія (створюються необхідні таблиці для роботи підсистеми управління репозиторієм патернів).

– Підключення до БД – підключення до бази даних (вибір пункту меню Підключення до БД).

– Вибір папки з БД – вибір папки з базою даних патернів.

– Завантаження з бази списку архітектур – обробка базою даних запиту на отримання списку архітектур.

– Вивід списку архітектур, що в базі – вивід підсистемою списку архітектур.

– Створення нової архітектури – процес запиту Адміністратора щодо імені та опису нової архітектури і подальше створення цієї архітектури в БД.

– Вибір архітектури – вибір Адміністратором архітектури на додання патерну в один з компонентів (натискання на ім’я архітектури і кнопки завантаження або подвійний клік на імені архітектури; увага: Разом з тим не збережена попередня архітектура буде замінена (тобто втрачена)).

– Запит структури архітектури з БД – створення запитів на збір інформації про архітектуру додатку.

– Збір структури архітектури – обробка запитів щодо збору інформації про архітектуру додатку.

– Відображення структури – відображення структури архітектури Адміністратору у відповідній зоні вікна підсистеми.

– Запит списку патернів, що належать компонентів – формування запиту на отримання списку патернів, що належать компонентів (категорія патернів).

– Завантаження списку патернів – зчитування списку патернів (обробка запиту).

– Вивід списку патернів, що належать компонентів – відображення вікна управління патернами та відображення відповідного списку патернів.

– Створення нового патерну – Адміністратор натискає кнопку створення нового патерну.

– Формування запиту на запис нового патерну в БД – підсистема формує запит до БД на запис патерну.

– Запис патерну – запис патерну в базу даних патернів.

– Внесення змін у патерн – Адміністратор записує зміни в патерн.

– Збереження змін в патерні – Адміністратор натискає кнопку збереження патерну в БД.

– Формування оновлення даних в БД – підсистема формує запит на оновлення інформації про патерн, включно з генерацією його попереднього перегляду.

– Оновлення даних в БД – внесення змін в БД патернів.

– Закриття підсистеми – користувач закриває підсистему управління репозиторієм патернів.

4 СПЕЦІАЛЬНА ЧАСТИНА

4.1 Адміністрування серверів Microsoft SQL Server за допомогою середовища SQL Server Management Studio

Середовище Microsoft SQL Server Management Studio є повнофункціональним інтегрованим адміністративним клієнтом, розробленим для вирішення завдань адміністратора сервера SQL Server. У середовищі Management Studio завдання адміністрування виконуються за допомогою оглядача об'єктів, який дозволяє підключитися до будь-якого сервера сімейства SQL Server і проглядати його вміст за допомогою графічних засобів. Сервер може бути екземпляром компоненту Database Engine, службами Analysis Services, службами Reporting Services, службами Integration Services або SQL Server Compact 3.5 SP1.

До числа засобів середовища Management Studio входять зареєстровані сервери, оглядач об'єктів, оглядач рішень, оглядач шаблонів, сторінка зведення і вікно документа. Щоб відобразити засіб, в меню "Вигляд" вибирається його назва. Для відображення редактора запитів слід натиснути кнопку "Створити запит" на панелі інструментів.

За замовчуванням мережевий трафік між Management Studio і SQL Server не шифрується.

Середовище Management Studio використовується для виконання наступних дій:

- реєстрації серверів;
- з'єднання з екземпляром компоненту Database Engine, службами SSAS, службами SSRS, службами SSIS або випуском SQL Server Compact 3.5 SP1;
- настройки властивостей сервера;
- управління об'єктами бази даних і службами SSAS, такими як куби, вимірювання і складки;

- створення таких об'єктів, як бази дані, таблиці, куби, користувачі бази даних і імена входу;
- управління файлами і групами файлів;
- приєднання або від'єднання баз даних;
- запуску засобів для роботи з сценаріями;
- управління безпекою;
- проглядання системних журналів;
- контролю поточної активності;
- настройки реплікації;
- управління повнотекстовими індексами.

Для запуску і зупинки SQL Server або агента SQL Server слід використовувати диспетчер конфігурації SQL Server.

Підключення до будь-якого компоненту сервера SQL Server з середовища SQL Server Management Studio

Середовище SQL Server Management Studio містить функції управління всіма компонентами SQL Server. Використовуйте середовище Management Studio для з'єднання з наступними компонентами і службами:

- екземпляром компоненту SQL Server Database Engine;
- Analysis Services;
- Integration Services;
- Reporting Services;
- SQL Server Compact 3.5 SP1.

Хоча середовище Management Studio дозволяє працювати із запитами без підключення до джерела даних, для виконання більшості інших завдань потрібне підключення. У середовищі Management Studio властивості з'єднання з компонентами SQL Server можна набудувати в діалоговому вікні З'єднання з сервером. При запуску Management Studio відкривається діалогове вікно З'єднання з сервером із запитом підключення до сервера. У діалоговому вікні З'єднання з сервером запам'ятовуються параметри, задані під час попереднього його використання.

4.2 Використання оглядача об'єктів

Оглядач об'єктів – компонент середовища SQL Server Management Studio – може з'єднуватися з екземплярами служб Database Engine, Analysis Services, Integration Services, Reporting Services і SQL Server Compact 3.5 SP1. Він забезпечує проглядання всіх об'єктів сервера і надає призначений для користувача інтерфейс для управління ними. Можливості оглядача об'єктів можуть трохи розрізнятися залежно від типу сервера, але в загальному випадку включають функції розробки для баз даних, а також функції управління для всіх типів серверів.

Оглядач об'єктів відображається в Management Studio за умовчанням. Якщо оглядач об'єктів не видно, відкрийте меню Вигляд і виберіть пункт Оглядач об'єктів.

Для використання оглядача об'єктів слід спочатку підключитися до сервера. На панелі інструментів оглядача об'єктів натисніть кнопку З'єднати і виберіть тип сервера із списку, що розкривається. Відкривається діалогове вікно З'єднання з сервером. Для підключення слід вказати принаймні ім'я сервера і правильні дані для перевірки достовірності.

При підключенні до сервера можна вказати додаткові дані з'єднання в діалоговому вікні З'єднання з сервером. У діалоговому вікні З'єднання з сервером зберігаються використані востаннє настройки, які використовуватимуться для нових з'єднань, наприклад для відкриття нових вікон редактора коди.

Щоб задати вибіркові настройки з'єднання, виконаються наступні операції.

На панелі інструментів оглядача об'єктів натисніть кнопку "З'єднати", а потім виберіть тип сервера, до якого проводиться з'єднання. Відкриється діалогове вікно "З'єднання з сервером".

У полі "Ім'я сервера" введіть ім'я екземпляра SQL Server.

Натисніть "Параметри". У діалоговому вікні "З'єднання з сервером" будуть показані додаткові параметри.

Перейдіть на вкладку "Властивості з'єднання" для настройки додаткових параметрів. Набір доступних параметрів залежить від типу сервера. Для компоненту Database Engine доступні наступні параметри (таблиця 1).

Таблиця 4.1 – Параметри компоненту Database Engine

Параметр	Опис
Підключитися до бази даних	Дозволяє вибрати одну з доступних баз даних на сервері. Список містить тільки ті бази даних, для перегляду яких є дозвіл.
Мережевий протокол	Дозволяє вибрати один з варіантів: "Загальна пам'ять", "TCP/IP" або "Іменовані канали".
Розмір мережевого пакету	Задається в байтах. Значення за замовчуванням рівне 4 096 байтам.
Час очікування з'єднання	Задається в секундах. Значення за умовчанням рівне 15 секундам.
Час очікування виконання	Задається в секундах. Значення за умовчанням (0) означає, що виконання ніколи не буде перервано із-за перевищення часу очікування.
Шифрування з'єднання	Задає примусове шифрування.

Щоб додати заданий сервер в список зареєстрованих серверів, перейдіть на вкладку "Зареєстрований сервер", потім клацніть позицію, де слід помістити сервер, і завершите з'єднання.

За допомогою сторінки "Додаткові параметри з'єднання" можна додати додаткові параметри з'єднання до рядка з'єднання.

При підключенні до сервера можна легко зареєструвати цей сервер для подальшого використання. У оглядачі об'єктів клацніть правою кнопкою миші ім'я сервера, а потім виберіть пункт "Зареєструвати". У діалоговому вікні

"Реєстрація сервера" вкажіть позицію в дереві груп серверів, в яку слід помістити сервер. У полі "Ім'я сервера" можна замінити ім'я сервера на більш описове. Наприклад, сервер APSQL02 можна зареєструвати із зрозумілішим ім'ям, допустимий, "Рахунки до оплати".

У оглядачі об'єктів для групування відомостей в теки використовується деревовидна структура. Для розкриття теки клацніть знак "плюс" (+) або двічі клацніть теку. При розкритті тек відображаються докладніші відомості. Для виконання стандартних завдань клацніть правою кнопкою миші теку або об'єкт. Для виконання найбільш часто виконуваного завдання клацніть об'єкт двічі.

При першому розкритті теки оглядач об'єктів запрошує на сервері дані для заповнення дерева. Під час заповнення дерева можна виконувати інші функції. Процес заповнення дерева в оглядачі об'єктів можна припинити, натиснувши кнопку "Зупинити". Подальші дії, наприклад фільтрація списку, будуть виконані тільки над вже заповненою частиною теки, поки вона не буде оновлена для заповнення наново.

З метою економії ресурсів за наявності багатьох об'єктів вміст тек дерева оглядача об'єктів не оновлюється автоматично. Щоб відновити список об'єктів в теці, клацніть цю теку правою кнопкою миші і виберіть пункт "Відновити".

У оглядачі об'єктів може відображатися до 65536 об'єктів. Після досягнення межі в 65536 видимих об'єктів переміщатися по додаткових об'єктах дерева оглядача об'єктів буде неможливо. Для перегляду додаткових об'єктів в оглядачі об'єктів закрийте невживані вузли або застосуйте фільтрацію, щоб зменшити число об'єктів.

Якщо тека містить велике число об'єктів, пошук потрібного об'єкту може бути утруднений. У таких випадках слід зменшити розмір списку, використовуючи функцію фільтрації оглядача об'єктів. Наприклад, хай потрібно знайти конкретного користувача бази даних або ж останню із створених таблиць в списках, що містять сотні об'єктів. Клацніть теку, до якої необхідно застосувати фільтрацію, а потім натисніть кнопку фільтрації, щоб відкрити діалогове вікно "Настройки фільтру". Фільтрацію списку можна здійснювати по імені, даті

створення, в деяких випадках – по схемі, а також можна вказувати додаткові оператори фільтрації, такі як Starts with, Contains і Between.

Одночасно в оглядачі об'єктів можна вибрати тільки один об'єкт. Щоб вибрати декілька об'єктів, натисніть клавішу F7, при цьому відкриється сторінка "Подробиці" оглядача об'єктів. Сторінка "Подробиці" оглядача об'єктів підтримує вибір декількох об'єктів.

Коли оглядач об'єктів підключений до сервера, то, використовуючи настройки з'єднання оглядача об'єктів, можна відкрити нове вікно "Редактор коду". Щоб відкрити нове вікно "Редактор коду", клацніть правою кнопкою миші ім'я сервера в оглядачі об'єктів, а потім виберіть пункт "Створити запит". Щоб відкрити вікно "Редактор коду" для конкретної бази даних, клацніть правою кнопкою миші ім'я бази даних, а потім виберіть пункт "Створити запит". При відкритті нового запиту до сервера служб Analysis Services можна вибирати запити розширень інтелектуального аналізу даних, багатовимірних виразів MDX або XML для аналітики.

Сеанс PowerShell можна запустити, клацнувши практично будь-яку з тек і об'єктів в дереві оглядача об'єктів правою кнопкою миші і вибравши пункт "Запуск PowerShell". В результаті цього запускається сеанс PowerShell з підтримкою SQL Server PowerShell, в якому встановлений шлях до об'єкту, вибраного клацанням правою кнопкою миші в оглядачі об'єктів. Після цього можна вводити команди PowerShell в інтерактивному середовищі PowerShell.

Щоб налаштувати параметри оглядача об'єктів, натисніть кнопку "Параметри" в меню "Засобу", а потім клацніть "Оглядач об'єктів SQL Server".

4.3 Довідка і підтримка користувачів в середовищі SQL Server Management Studio

Довідка і підтримка користувачів в середовищі SQL Server Management Studio доступна через меню "Довідка" і в електронній документації SQL Server. У меню "Довідка" в Management Studio пропонується декілька варіантів доступу до відомостей про SQL Server. Воно також надає доступ до співтовариства SQL

Server і ресурсів інтерактивної бібліотеки MSDN, раніше недоступним з довідкового середовища. Довідкове середовище тепер також може бути налаштоване як на запуск з середовища SQL Server Management Studio, так і у власному зовнішньому вікні.

Вкладки "Зміст" і "Індекс" надають функції і інтерфейс, вже знайомі користувачам SQL Server. Доступні наступні параметри.

1. Інструкції.

Надає ієрархічний набір зв'язаних сторінок з корисними статтями, що описують типові завдання при роботі з SQL Server. Вміст впорядкований по компонентах і завданнях, наприклад розділ "Реплікації" і так далі

4. Пошук.

Виконується пошук по розділах із зумовленими фільтрами або без них. Пошук в SQL Server – це окрема сторінка з вкладками. Користувачі можуть уточнювати пошук, встановлюючи фільтри за певним типом розділу, мовою або технологією. За умовчанням при пошуку не використовуються ніяких настроєних фільтрів, при цьому пошук виконується тільки по встановлених колекціях.

3. Динамічна довідка.

Автоматично відображає посилання на потрібні відомості, поки користувачі працюють в середовищі Management Studio.

4. Вибрані розділи довідки.

Зберігаються призначені для користувача закладки для полегшення повторного доступу.

Інтерактивна бібліотека MSDN Online і співтовариства SQL Server

Довідка в Management Studio також надає користувачам можливості підключення до MSDN Online і тематичних співтовариств по SQL Server в Інтернеті для отримання відомостей. Можна:

- дістати доступ до співтовариств SQL Server із сторінки "Інструкції";
- виконати пошук в MSDN Online і веб-вузлах співтовариства SQL Server.

Довідку в SQL Server Management Studio можна викликати двома способами. За замовчанням, якщо електронна документація SQL Server відкривається з Management Studio, вона відкривається у вікні документа, зовнішньому щодо середовища Management Studio. Це вікно все ж таки пов'язане з Management Studio: воно може реагувати на деякі події Management Studio, а при закритті Management Studio воно теж закриється. Відкривати таким чином електронну документацію особливо зручно за наявності двох моніторів: вікно електронної документації можна перетягнути на другий монітор, щоб воно не заважало працювати в першому вікні, але було легко доступно для перегляду.

Можна також відкрити електронну документацію у вікні документа усередині Management Studio. Це зручно при обмеженому просторі екрану; в цьому випадку корисно використовувати можливість приховувати вікна Management Studio.

Якщо необхідно, щоб електронна документація була повністю незалежна від Management Studio, відкрийте електронну документацію SQL Server з меню "Пуск", тоді вона не реагуватиме на дії в середовищі Management Studio і не закриється при виході з Management Studio.

4.4 Можливості редакторів

Редактори коди в середовищі SQL Server Management Studio володіють наступними можливостями.

1. Шаблони, які можуть бути використані для швидкої підготовки сценаріїв для компоненту SQL Server Database Engine, служб Analysis Services і SQL Server Compact 3.5 SP1. Шаблони – це файли, що містять базовий набір інструкцій, необхідних для створення об'єктів в базі даних.

2. Виділення кольором синтаксичних конструкцій, що полегшує читаність складних інструкцій.

3. Створення запитів в графічному конструкторі запитів методом перетягання.

4. Представлення вікон запитів у вигляді вкладок вікна документа або у вигляді окремих документів.

5. Представлення результатів виконання запиту у вигляді табличної сітки або текстового вікна з можливістю перенаправлення у файл.

6. Відображення табличної сітки результатів у вигляді окремих вікон з вкладками.

7. Графічне відображення даних інструкції Showplan з демонстрацією логічних кроків, вбудованих в план виконання інструкції мови TRANSACT-SQL.

Середовище зміни тексту з розвиненими можливостями, що підтримує пошук і заміну, коментування блоків, призначені для користувача шрифти і кольори і нумерацію рядків. Деякі типи редакторів підтримують додаткові можливості, такі як структуризація і автозавершення.

Режим SQLCMD для виконання сценаріїв, що містять команди операційної системи.

При виборі цього параметра слід враховувати наступні обмеження.

Технологія IntelliSense відключена в редакторові запитів компоненту Database Engine.

Оскільки редактор запитів не запускається з командного рядка, йому не можна передавати параметри командного рядка, такі як змінні.

Оскільки редактор запитів не може відповідати на запити операційної системи, необхідно дотримуватися обережності і не запускати інтерактивні інструкції.

Включення технології IntelliSense для сценаріїв TRANSACT-SQL великого розміру може привести до падіння продуктивності на повільних комп'ютерах.

Редактори запитів містять наступні вікна.

1. Редактор запитів. Це вікно використовується для введення і виконання сценаріїв.

2. Результати. Це вікно використовується для проглядання результатів виконання запиту. Результати в нім можуть відображатися у вигляді тексту або табличної сітки.

3. Повідомлення. У цьому вікні відображаються помилки, попередження і інформаційні повідомлення, повертані сервером в ході виконання сценарію. Список повідомлень змінюється тільки при повторному запуску сценарію.

4. Список помилок. У цьому вікні відображаються синтаксичні і семантичні помилки, виявлені функцією IntelliSense в редакторові запитів Database Engine. Список помилок міняється динамічно в ході редагування сценаріїв мови TRANSACT-SQL. Список помилок відображає помилки тільки в редакторові запитів Database Engine; у інших редакторах список помилок не відображається.

5. Статистика клієнта. У цьому вікні відображаються зведення про виконання запиту, згруповані по категоріях. При виборі пункту Включити статистику клієнта з меню Запит в ході виконання запиту з'являється вікно Статистика клієнта. Статистика успішно виконаних запитів приводиться разом з середніми значеннями. Щоб скинути середні значення, виберіть пункт Скинути статистику клієнта в меню Запит.

У таблиці 4.2 перераховані розділи електронної документації SQL Server, що містять зведення про редактора коду.

4.5 Конструктори візуальних інструментів для баз даних

Візуальні інструменти для баз даних є поєднанням засобів конструювання, які можна використовувати для роботи з джерелами даних. З їх допомогою можна створювати запити, конструювати і міняти структуру бази даних і оновлювати дані. До числа цих засобів входять конструктор схем баз даних, конструктор таблиць і конструктор запитів і уявлень.

4.5.1 Конструктор схем баз даних

Конструктор схем баз даних відкриває вікно, де можна візуально створювати і змінювати таблиці і зв'язки в базі даних, а також проглядати їх.

Таблиця 4.2 – Розділи електронної документації SQL Server

Розділ	Опис
Використання шаблонів в середовищі SQL Server Management Studio	Містить відомості про шаблони і їх створення.
Команди і функції для підвищення зручності роботи в редакторах	Включає теми про такі можливості, як структуризація коди, нумерація рядків, гіперпосилання в коментарях, позначки, перенос по словах і тому подібне
Поєднання клавіш середовища SQL Server Management Studio	Містить перелік поєднань клавіш, доступних в редакторові коди.
Як пов'язати розширення файлу з редактором коди	Пояснює порядок настройки середовища SQL Server Management Studio для відкриття потрібного редактора коди залежно від розширення файлу.
Зміна сценаріїв і файлів в середовищі SQL Server Management Studio	Містить посилання на сторінки, що описують багато параметрів настройки редактора коди.
Як змінити колір, розмір і стиль шрифту	Описує настройку зовнішнього вигляду тексту в редакторові коду.
Вікно "Список помилок" (середовище Management Studio)	Описує відображення відомостей про помилки в редакторові запитів Database Engine.

Щоб відкрити конструктор схем баз даних, відкрийте вже існуючу схему або в оглядачі об'єктів клацніть правою кнопкою миші вузол "База даних" і в списку, що розкривається, виберіть Створити схему бази даних.

Після відкриття конструктора в головному меню з'явиться меню Схеми бази даних. Це меню є точкою доступу до спеціальних можливостей конструктора.

Конструктор працює з базами даних Microsoft SQL Server.

Дана версія візуальних інструментів для баз даних не підтримує Microsoft SQL Server версії 7 і раніші версії.

4.5.2 Конструктор таблиць

Конструктор таблиць – візуальний засіб проектування і візуалізації окремих таблиць бази даних Microsoft SQL Server, до якої підключений користувач.

Вікно конструктора ділиться на дві області. У верхній області показана таблиця, в кожному рядку якої описується один стовпець таблиці бази даних. Для кожного стовпця відображаються його істотні атрибути: ім'я стовпця, тип даних і чи дозволені значення NULL.

У нижній області конструктора таблиць на вкладці "Властивості стовпця" відображаються додаткові атрибути будь-якого вибраного у верхній таблиці стовпця.

Клацнувши правою кнопкою миші в таблиці конструктора можна дістати доступ до діалогових вікон, в яких можна створювати і міняти зв'язки, обмеження, індекси і ключі таблиці бази даних.

Щоб відкрити конструктор таблиць, відкрийте вже існуючу таблицю або клацніть правою кнопкою миші вузол Таблиці в оглядачі об'єктів і в списку, що розкривається, виберіть Додати нову таблицю.

Після відкриття конструктора в головному меню з'явиться меню "Конструктор таблиць". Це меню є точкою доступу до спеціальних можливостей конструктора.

Конструктор працює з базами даних Microsoft SQL Server.

Дана версія візуальних інструментів для баз даних не підтримує Microsoft SQL Server версії 7 і раніші версії.

4.5.3 Конструктор запитів і представлень

Конструктор запитів і представлень фактично є двома засобами, що працюють схожим чином. До деяких з їх основних відмінностей відноситься наступне.

Представлення зберігаються в базі даних, а запити зберігаються в проекті бази даних середовища Visual Studio.

Конструктор запитів працює практично з будь-якими джерелами даних, а конструктор представлень підтримує тільки SQL Server.

Конструктор запитів дозволяє проектувати інструкції мови маніпулювання даними SELECT, INSERT, UPDATE і DELETE, а уявлення можуть містити тільки інструкції SELECT.

Конструктор представлень дозволяє проектувати і наочно відображати існуючі уявлення або створювати нові в базі даних Microsoft SQL Server, до якої підключений користувач.

Вікно конструктора ділиться на чотири області: область схем, область критеріїв, область "SQL" і область результатів. Додаткові відомості про кожну з цих областей див. в розділі Інструменти конструктора запитів і уявлень (візуальні інструменти для баз даних).

Щоб відкрити Конструктор представлень, відкрийте вже існуюче уявлення або клацніть правою кнопкою миші вузол Уявлення в оглядачі об'єктів і в списку, що розкривається, виберіть Додати нове уявлення.

Після відкриття конструктора в головному меню з'явиться меню Конструктор запитів. Це меню є точкою доступу до спеціальних можливостей конструктора.

Конструктор працює з базами даних Microsoft SQL Server.

Дана версія візуальних інструментів для баз даних не підтримує Microsoft SQL Server версії 7 і раніші версії.

4.5.4 Конструювання схем баз даних

Конструктор баз даних – це візуальний засіб, що дозволяє конструювати і візуалізувати базу даних, з якою встановлено з'єднання. При конструюванні за

допомогою конструктора баз даних можна створювати, редагувати або видаляти таблиці, стовпці, ключі, індекси, зв'язки і обмеження. Для візуалізації бази даних можна створити одну або декілька діаграм, що ілюструють деякі або все наявні в ній таблиці, стовпці, ключі і зв'язки (рисунок 4.1).

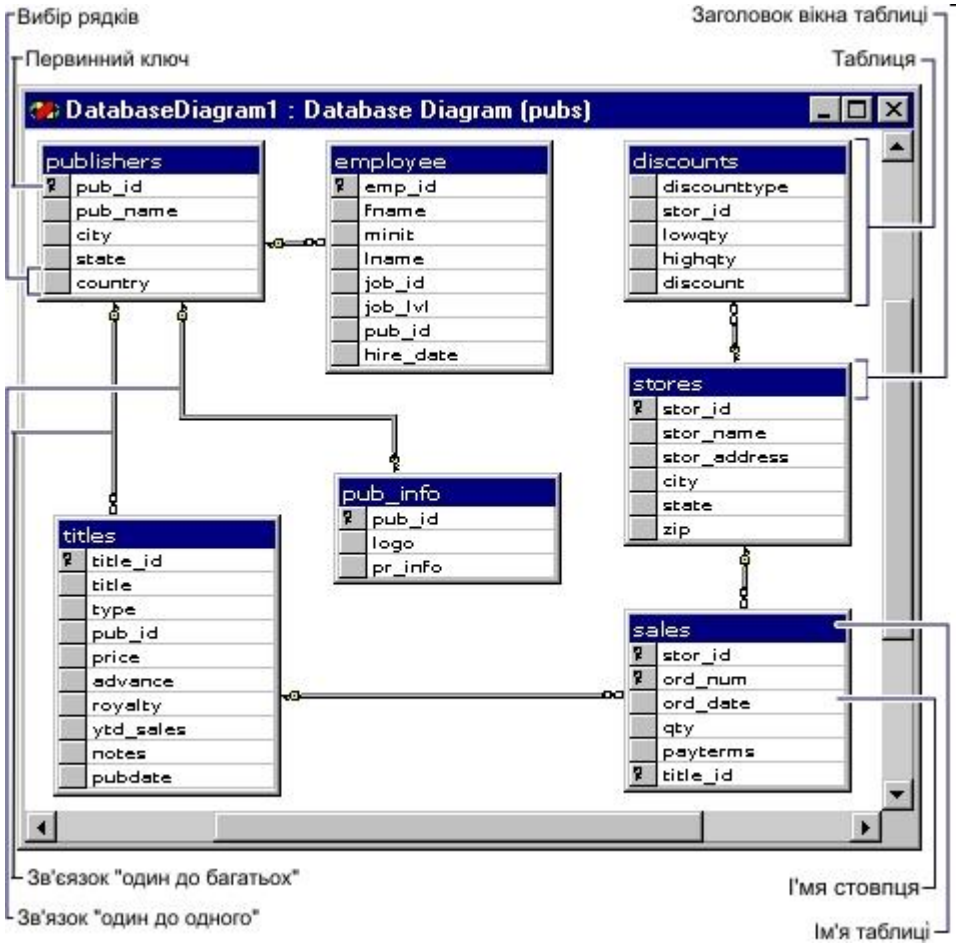


Рисунок 4.1 – Вікно конструктора схеми бази даних

Для будь-якої бази даних можна створити будь-яку необхідну кількість схем; кожна з таблиць бази даних може використовуватися в будь-якій кількості схем. Таким чином, для візуалізації різних частин бази даних або для акцентування різних аспектів її конструювання можна створювати різні схеми. Наприклад, можна створити велику схему, в якій відображатимуться всі таблиці і стовпці, а також меншу схему, в якій відображатимуться всі таблиці, але не буде стовпців.

Кожна створена схема бази даних зберігається у відповідній базі даних.

Всередині схеми бази даних кожна таблиця має три окремі елементи: рядок заголовка, список вибору рядків і набір стовпців властивостей. У рядку заголовка відображається ім'я таблиці.

Якщо таблиця була змінена, але ще не збережена, то після імені таблиці з'являється зірочка (*), що показує наявність незбережених змін.

Набір стовпців властивостей видно не у всіх представленнях таблиці. Таблицю можна проглянути в будь-якому з п'яти різних уявлень, що дозволяють підібрати відповідний розмір і розміщення елементів діаграми.

Всередині схеми бази даних у кожного зв'язку є три окремі елементи: кінцеві точки, стиль лінії і зв'язані таблиці.

Кінцеві точки лінії показують вид зв'язку: "один до одного" або "один до багатьох". Якщо на одній кінцевій точці зв'язку знаходиться ключ, а на іншій – цифра вісім, то це зв'язок "один до багатьох". Якщо у зв'язку по одному ключу на кожній кінцевій крапці, то це зв'язок "один до одного".

Стиль лінії (не її кінцеві точки) показує, чи перевіряє СУБД посилальну цілісність для зв'язку при додаванні нових даних в таблицю, зв'язану за допомогою зовнішнього ключа. Якщо зв'язок намальований у вигляді суцільної лінії, це означає, що СУБД перевіряє посилальну цілісність для зв'язку при додаванні або зміні рядків в таблиці, зв'язаній за допомогою зовнішнього ключа. Якщо лінія пунктирна, це означає, що СУБД не перевіряє посилальну цілісність для зв'язку при додаванні або зміні рядків в таблиці, зв'язаній за допомогою зовнішнього ключа.

Лінія зв'язку показує, що дві таблиці зв'язано за допомогою зовнішнього ключа. Для зв'язку "один до багатьох" таблиця, зв'язана за допомогою зовнішнього ключа, – це таблиця біля цифри 8 на лінії. Якщо обидві кінцеві точки лінії приєднано до однієї таблиці, це означає поворотний зв'язок.

5 ОБҐРУНТУВАННЯ ЕКОНОМІЧНОЇ ЕФЕКТИВНОСТІ

Встановлення економічної доцільності проведення розробки є основною метою виконання даного розділу дипломної роботи.

Метою роботи є розробка програмного забезпечення для аналітичного аналізу текстових даних. Доступ до бази даних організовано без застосування спеціалізованого програмного забезпечення.

5.1 Розрахунок норм часу на виконання науково-дослідної роботи

Ефективне використання часу має велике значення тому, що коефіцієнт корисної дії залежить від оптимального використання часу.

Для полегшення і структуризації виконання розробки, її поділяють на етапи.

Основні етапи при виконанні НДР (розробки програми):

- постановка проблеми;
- аналіз виконаних досліджень;
- проектування бази даних;
- розробка програми;
- створення вихідних документів;
- оформлення документації.

Для оцінки тривалості виконання окремих робіт використовують нормативи часу або попередній досвід.

До таких нормативів відносять тривалість написання операції (команд), яка на деяких підприємствах становить:

- 1 -ї операції – (0,5–1,6 год)
- 5-ти операцій – 8 год. (тривалість зміни).

У разі їх відсутності звертаються до експертних оцінок по встановленню тривалості кожного етапу (стадії):

при трьох оцінках:

$$T_{вс}=(t_{min}+4t_{н.й}+t_{max})/6, \quad (5.1)$$

при двох оцінках:

$$T_{вс}=(3t_{min}+2_{tmax})/5, \quad (5.2)$$

де $T_{вс}$ – очікуване (середнє) значення тривалості виконання етапу (стадії); t_{min}, $t_{н.й}$, t_{max} – відповідно мінімальна, найбільш імовірна і максимальна оцінки тривалості виконання етапу (стадії). Для визначення загальної тривалості проведення НДР (розробки програми) доцільно дані витрат часу на виконання окремих стадій (етапів) звести у таблицю 5.1.

Середній час виконання НДР (розробки програми)

Таблиця 5.1 – Основні етапи і час їх виконання у НДР

Номер етапу	Середній час виконання стадії (етапу), год.	
	Інженер	Керівник
Постановка завдання	58	5
Аналіз виконаних досліджень	104	10
Проектування бази даних	120	11
Розробка програми	80	5
Створення вихідних документів	50	3
Оформлення документації	95	7
Разом	508	41

Витрати часу керівника на виконання окремих стадій (етапів) при недостатній кількості інформації доцільно приймати в межах 5% сумарних витрат часу інженерів на виконання цих стадій (етапів).

5.2 Розрахунок витрат на проведення НДР

Розрахунок поточних витрат на проведення НДР (розробки програми) проводять в розрізі таких калькуляційних статей:

- основна заробітна плата (з/п);

- додаткова (з/п);
- нарахування на (з/п);
- косультаційні витрати;
- матеріали для виконання робіт по НДР;
- експериментально-виробничі витрати;
- загальновиробничі витрати;
- адміністративні витрати;
- позавиробничі витрати.

Системна обробка інформації щодо обліку праці і заробітної плати є однією з найрізноманітніших, складних і трудомістких ділянок роботи. Це і зрозуміло. Як показують дослідження за трудомісткістю ця ділянка становить майже одну третину від всього обліку по підприємству.

Реалізація на практиці процесу справедливого розподілення матеріальних благ. А також пов'язаний з цим контроль цілком і повністю залежить від вірної організації (на науковій основі) праці і достовірного нарахування заробітної плати. Без цього навряд чи можна установити необхідний контроль за мірою праці і мірою винагородження за неї, за виконання виробничого плану, плану по праці і заробітній платі, а також за рівнем і зростом їх продуктивності.

При системному розв'язанні питання про облік праці і заробітної плати велике значення має умовно-постійна (нормативна, довідкова та інша) інформація, яка в даному разі характеризує переважно постійних виконавців (людей і механізми) та постійні процеси (технологічні операції). Тому у першу чергу зміст по обліку праці і заробітної плати неодмінно повинна входити інформація про виконавців (облік складу працівників). Для правильної оплати потрібні достовірний і своєчасний облік виробітку та визначення його якості. Облік виробітку і його оплати – найбільш трудомістка ділянка роботи, вона складається з ряду проміжних операцій.

Застосування ПЕОМ у вигляді АРМ, а також широке використання умовно-постійної інформації значно поліпшує облік показників про виробіток безпосередньо відноситься до обліку результатів і одночасно ці ж данні

використовуються для нарахування відрядної заробітної плати. Так здійснюється зв'язок між цими ділянками обліку, а також взаємоконтроль цих показників.

Основна з/п складається із прямої з/п і доплати, яка при укрупнених розрахунках становить 25% – 35% від прямої з/п. При розрахунку з/п кількість робочих днів в місяці слід приймати – 25,4 дні/міс., що відповідає 203,2 год./міс. Приймемо розмір місячних окладів керівника 1600 грн. та інженерів 1420 грн. згідно існуючих на даний час норм. Пряма з/п визначається:

$$ЗП = (О_i \cdot Т_i)/203,2, \quad (5.3)$$

де $О_i$ – розмір місячних окладів і-х категорій працівників;

$Т_i$ – трудомісткість робіт виконаних працівниками і-х категорій.

Для інженера: ЗП = (1420*508)/203,2=3543,01 грн.;

Для керівника: ЗП = (1600*41)/203,2=314,96 грн.;

Величина доплат:

$$ЗП_і=ЗП \cdot К_і \quad (5.4)$$

де $К_і$ – коефіцієнт доплат (0,25 – 0,35).

Вибираємо коефіцієнт 0,28.

Для інженера: $ЗП_і$=3543,01 *0,28=992,04грн.

Для керівника: $ЗП_і$=314,96*0,28=88,19 грн.

Основна з/п: $ЗП = ЗП + ЗП_і$.

Для інженера: $ЗП_о$ = 3543,01 +992,04=4535,05 грн.

Для керівника: $ЗП_о$ =314,96+88,19 =403,15 грн.

Величина додаткової з/п

$$ЗП_Д = ЗП_о \cdot К_Д, \quad (5.5)$$

де $К_Д$ – коефіцієнт додаткової з/п (0,05 – 0,1).

Нехай $К_Д$=0,1, тоді:

$ЗП_Д$ = 4535,05 *0,1=453,51 грн.

$ЗП_Д$ =403,15 *0,1=40,32 грн.

для інженера і керівника відповідно.

Витрати, на проведення НДР (розробки програми) крім річного фонду заробітної плати включають ще й соціальні нарахування. Нормативи нарахувань на заробітну плату інженера наступні:

Всього норматив нарахувань на заробітну плату інженера становить

38% – (ЗПо+$ЗП_Д$)·0,38=4938,20*0,38=1876,52 грн.

Для керівника: 443,47*0,38=168,52 грн.

Результати обрахунків по з/п занесемо в таблицю 5.2.

Таблиця 5.2 – Зведена відомість витрат на заробітну плату, грн.

Категорія працівників	Основна заробітна плата			Додаткова заробітна плата	Нарахування на заробітну плату	Всього витрати на заробітну плату
	Пряма заробітна плата	Доплати	Всього			
Інженер	3543,01	992,04	4535,05	453,51	1876,52	6865,08
Керівник	314,96	88,19	403,15	453,51	1876,52	2733,18
Всього	3857,97	1080,23	4938,20	907,02	3753,04	9598,26

При необхідності, проводячи організаційно-економічне обґрунтування ДП, слід враховувати витрати на консультації. Такі витрати можна врахувати окремою калькуляційною статтею виходячи з реальних цін на певний вид консультаційних послуг. Як правило, при отриманні консультацій витрати рахуються на оплату праці консультантів за певний консультаційний час.

Для розрахунку витрат на консультації, врахуємо, що консультації були надані в обсязі 15 год., вартість їх 75 грн.

Витрати на матеріали розраховуються на основі норм їх витрат і відповідних оптових цін:

$$М_з = \sum_{i=1}^{n} Н_{мі} \cdot Ц_{оі} \quad , \qquad (5.6)$$

де $М_з$ – затрати на матеріали;

$Н_{мі}$ – норма затрат і-х матеріалів;

$Ц_{оі}$ – оптова ціна за одиницю витрат і-х матеріалів;

n – кількість найменувань матеріалів.

До одержаного результату додамо транспортно-заготівельні затрати на рівні 6% їх прейскурантної вартості.

Результати розрахунку затрат на матеріали зводяться в таблицю 5.3.

Таблиця 5.3 – Визначення величини затрат на матеріали

Найменування матеріальних ресурсів	Одиниця виміру	Норма витрат	Ціна за одиницю, грн.	Затрати матеріалів, грн.	Транспортно-заготівельні витрати, грн.	Загальна сума витрат на матеріали, грн.
1. Основні матеріали						
Принтер	штука	1	1950,00	1950,00	117,00	2067,00
Комп’ютер	штука	1	5000,00	5000,00	300,00	5300,00
Ліцензії на антивірусне програмне забезпечення	штука	1	160,00	160,00	9,60	169,60
Канцтовари	штука	64	1,00	64,00	3,84	67,84
2. Допоміжні матеріали						
Папір	Упаковка	4	30,00	150,00	9,00	159,00
Інсталяційні диски	Штук	8	3,00	15,00	0,90	15,90
Разом						7779,30

Експериментально-виробничі витрати визначаються як витрати на машинний час, який є потрібним для виконання необхідного об'єму робіт виходячи з його вартості за одиницю часу. Вартість роботи на ПЕОМ і користуванням мережею Інтернет встановлюємо виходячи з реальних даних (при укрупнених розрахунках можна прийняти тариф роботи на ПЕОМ 4 грн./год.).

Загальновиробничі витрати при укрупнених розрахунках приймаємо на рівні 70% – 90% від суми основної і додаткової з/п інженерів, яка була нарахована за роботу по проведенні НДР (розробки мережі). Аналогічно визначаються адміністративні витрати, які доцільно приймати на рівні 50% –

60% від суми основної і додаткової з/п інженерів. Позавиробничі витрати слід приймати на рівні 3% – 7% від виробничої собівартості.

Розрахунок поточних витрат на проведення НДР (розробки програмного продукту) зводиться в таблицю 5.4.

Заключною частиною роботи на цій ділянці є показники, що необхідні для встановлення собівартості, проведення комплексного економічного аналізу затрат праці і нарахованої заробітної плати.

Таблиця 5.4 – Калькуляція собівартості проведення НДР (розробки програми)

Статті витрат	Витрати, грн.	В відсотках до загальної суми
1 . Основна заробітна плата	4938,20	17,45
2. Додаткова заробітна плата	907,02	3,20
3. Нарахування на заробітну плату	3753,04	13,26
4. Консультації	75,00	0,26
5. Матеріали	7779,30	27,49
6. Експериментально-виробничі витрати	2188,00	7,73
7. Загальновиробничі витрати	4100,00	14,49
Разом виробнича собівартість	23740,56	83,88
8. Адміністративні витрати	2900,00	10,25
9. Позавиробничі витрати	1661,84	5,87
Повна собівартість	28302,40	100,00

Таким чином для розрахунку ціни НДР, необхідно забезпечити системний обліку праці і заробітної плати, що забезпечує:

– облік складу і рухів працівників при розробці програми;

– табельний облік робітників і службовців, а також облік використання робочого часу у різних аспектах.

5.3 Розрахунок ціни НДР і економічна ефективність від використання програми

5.3.1 Встановлення вартості

Ціна як економічна категорія завжди посідала особливе місце у виробничій діяльності підприємства. Очевидним є те, що в ринкових умовах в ціні перетинаються економічні інтереси виробників і споживачів. Ціна це грошовий вираз вартості товару. Вона відображає його споживчу корисність в конкретних ринкових умовах. Ринкова ціна забезпечує динамічну рівновагу між попитом та пропозицією, між суспільною вартістю товару і її грошовим виразом. Встановлення ціни на доцільному рівні, а також прогнозування динаміки цін світового ринку неможливо без врахування основних ціноутворюючих факторів. Їх можна розділити на три групи:

Ціну НДР (розробки програмного продукту) можна визначити

$$Ц = (С_{пр}/N_з + С_{коп}) + П, \quad (5.7)$$

де $С_{пр}$ – собівартість НДР (розробки програмного продукту), грн.;

$N_з$ – кількість замовлень (в нашому випадку 1 замовлення), од.;

$С_{коп}$ – собівартість копіювання (ксерокопії, дискети, компакт-диски, поштові витрати, відрядження спеціалістів для запуску та наладки програмного і апаратного забезпечення тощо), грн.

П – нормативна величина прибутку (15% - 30% від собівартості $С_{пр}$).

Таким чином, ціна мережі
Ц = (28302,40/1+2362,9)+0,2*28302,40=36325,78 грн.

5.3.2 Оцінка економічної ефективності розробки НДР (розробки програми)

Економічна ефективність від використання НДР (програмного забезпечення) зумовлена:

- скороченням трудовитрат при виконанні певних завдань;
- скороченням машинного часу при виконанні певних завдань.

При визначенні економічної ефективності необхідно порівняти використовуваний (базовий) продукт і пропонований. З допомогою відповідних розрахунків (в разі значної складності використання експертних оцінок)

визначається скорочення трудовитрат і (або) машинного часу, і як наслідок – економія коштів при використанні мережі.

Для визначення ефективності продукту розраховують чисту приведену цінність (чистий приведений прибуток) NPV і термін окупності $Т_{ок}$

$$NPV = \Sigma ((Д_t – В_t)/(1 + i)^t), \quad (5.8)$$

де $Д_t$ – повний дохід за рік t при використанні мережі;

$В_t$ – повні витрати за рік t при використанні мережі;

t – відповідний рік проекту;

i – дисконтна ставка (0,3).

Отже, розрахуємо чистий приведений прибуток. Методом експертних оцінок визначимо, що витрати на експлуатацію мережі становитимуть 950 грн. За перший рік експлуатації і по 700 грн. за наступні роки.

$NPV=(5660,48-950 / (1+0,3)^1) + (5660,48-700 / (1+0,3)^2) + (5660,48-700 / (1+0,3)^3)= 15517,85$ грн.

Період окупності

$$Т_{ок} = Ц/\Sigma ((Д_t – В_t)/(1 + i)^t). \quad (5.9)$$

Для програмних продуктів доцільним вважається термін окупності, який не перевищує 1 – 3 роки в залежності від функціонального призначення. Отже, $Т_{ок} = 36325,78/15517,85 = 2,3$ роки.

Результати розрахунків економічної ефективності проектних рішень вносяться в таблицю 5.5.

Таблиця 5.5 – Техніко-економічні показники порівняльних варіантів за розрахунковий період

№ п/п	Назва показника	Один. вимір.	Величина
1.	Витрати часу на розробку програмного продукту	год.	547

2.	Витрати на розробку програмного продукту	грн.	28302,40
3.	Кількість покупців програмного продукту	од.	1
4.	Ціна програмного продукту	грн.	36325,78
5.	Чиста приведена цінність програмного продукту	грн.	15517,85
6.	Термін окупності витрат по проекту	рік	2,3

Таким чином, за результатами розрахунків розроблена програма може бути впроваджена на підприємство.

6 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

6.1 Загальні закономірності виникнення небезпек

Соціальний і політичний стан країни формує відповідне суспільство і відносини в ньому. Безпека життєдіяльності впливає на стан суспільства як результат дії відповідних чинників життєдіяльності. Коли ці чинники вносять негативні дії в життя суспільства, то вони виступають в ролі небезпек. Суть небезпек, які можуть бути соціальними, політичними та ін., полягає в тому, що вони відображають, як проблеми безпеки життєдіяльності адаптуються в суспільстві.

Отже, чинник небезпеки з одного боку характеризує стан суспільства, а з іншого, є змістом причини його переходу в стан з негативними наслідками.

За першим змістом чинник формує можливі шляхи впливу на суспільство (чи його частину). Фактично це можливість здійснення управління чинником, а через нього – і управління суспільством з відповідних мотиваційних позицій.

За другим змістом має місце визначення чинника в ролі небезпеки, що потребує розробки передумов запобігання негативним наслідкам. Це зумовлює необхідність розробки механізму втручання в стан суспільства (теж є своєрідне управління), але через дуже складні можливі наслідки такий механізм треба розглядати як незалежний від впливу (а може і навіть за змістом як джерело мотивації) в межах надзвичайних ситуацій, що має місце в розвитку суспільства.

Характер формування соціальних і політичних небезпек, спричинених проблемами безпеки життєдіяльності наведено на рис. 6.1. Вплив діяльності людини на суспільство здійснюється через чинники впливу, що формують зміни (в суспільстві) та вищезгадані небезпеки. Склад передумов, що створюють чинники, подано також на рис. 6.1 а), б), в).

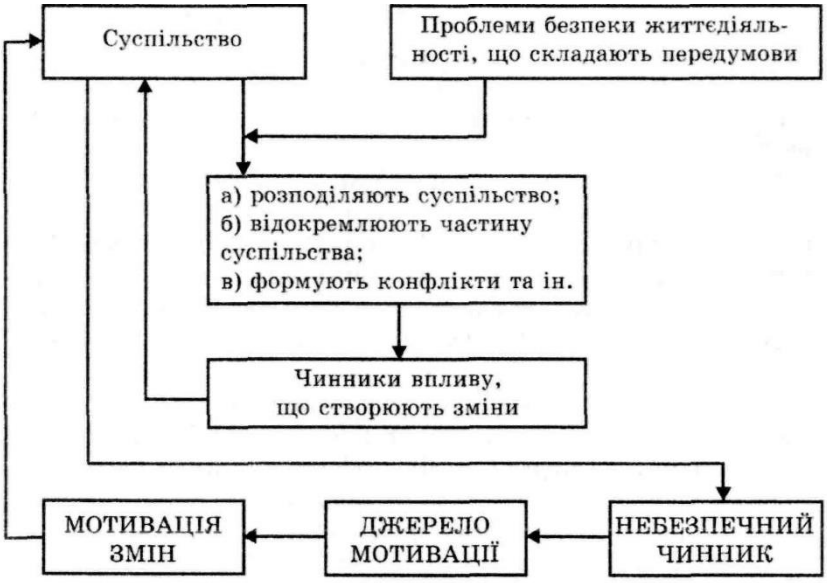


Рисунок 6.1 – Вплив життєдіяльності на стан суспільства

Таким чином, крім визначених небезпек (щодо безпеки життєдіяльності), існують також соціальні і політичні небезпечні чинники (див. рис. 6.2).

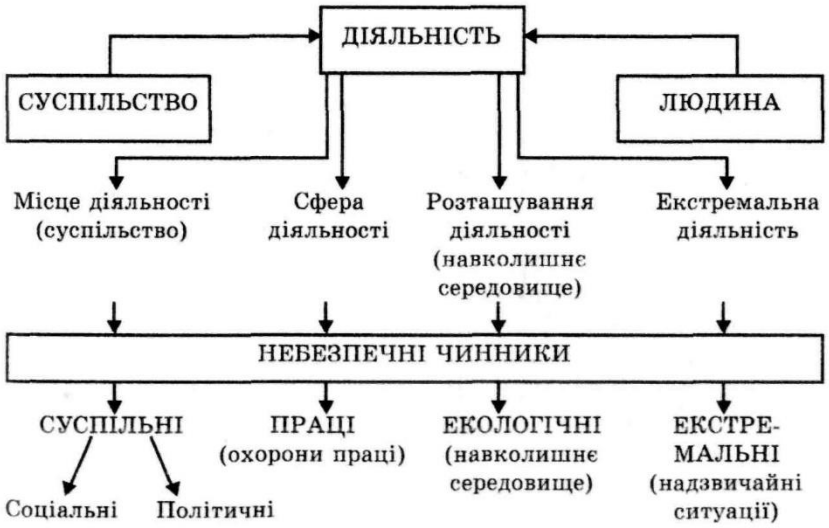


Рисунок 6.2 – Склад небезпечних чинників, що формують стан суспільства

Соціальні і політичні небезпеки, спричинені проблемами життєдіяльності, мають дуже складну природу. Суть цієї природи складає накопичення наслідків постійно діючих конфліктних, стресових та інших негативних ситуацій. У більшості випадків небезпека виражається у відношенні до чисельності суспільства у цілому кількості людей, що беруть участь у конфліктах і відокремлені від суспільства.

У цьому разі суть небезпеки має прояви не тільки у певному співвідношенні. Небезпека також існує у вигляді причини, що виникає в разі розподілу суспільства (причини конфлікту).

Види небезпек суспільства (соціальні і політичні небезпеки, що спричинені проблемами безпеки життєдіяльності):

1. Наявність конфлікту, що може перейти в негативні для суспільства наслідки в межах проблем безпеки життєдіяльності:

– наявність екологічної деградації середовища;

– загроза деградації продукції сільськогосподарського виробництва (продукції харчування);

– існування деградації самого суспільства;

– наявність (чи можливість) воєнного конфлікту;

– деградація національних відносин та ін.

2. Загроза розподілу суспільство – людина:

– існування протиріч суспільство – людина;

– наявність протиріч суспільство – натовп;

– існування протиріч суспільство – партія;

– існування протиріч суспільство – система управління та ін.

3. Загроза виникнення конфлікту між окремими групами (колективами) суспільства:

– керівництво – колектив;

– колектив – людина;

– мешканці – мешканці;

– колектив – колектив;

– «юрба» – «юрба».

Певні види небезпек створюють передумови для комплексної системи їх вивчення: природи, виникнення, існування, дії (на людину). В загальному вигляді це наведено на рис. 6.3, де людина є об'єктом впливів, які формують характер її стану.

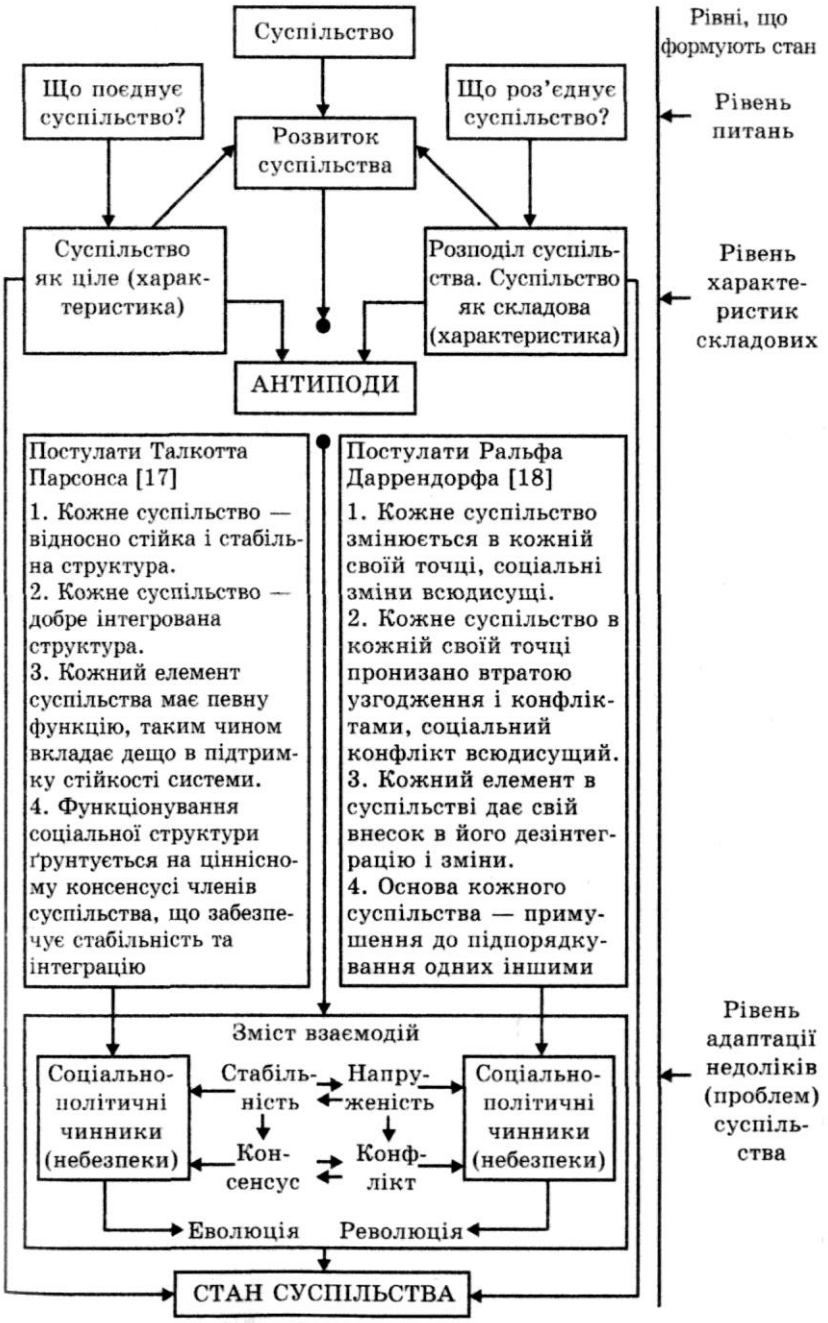


Рисунок 6.3 – Шляхи формування конфліктів і змін суспільства

Існування соціальних небезпек можна відстежити на конкретних прикладах розвитку негараздів. На рис. 6.5 а) наведено приклад можливого стану суспільства в умовах дії закону «на всіх всього не вистачить».

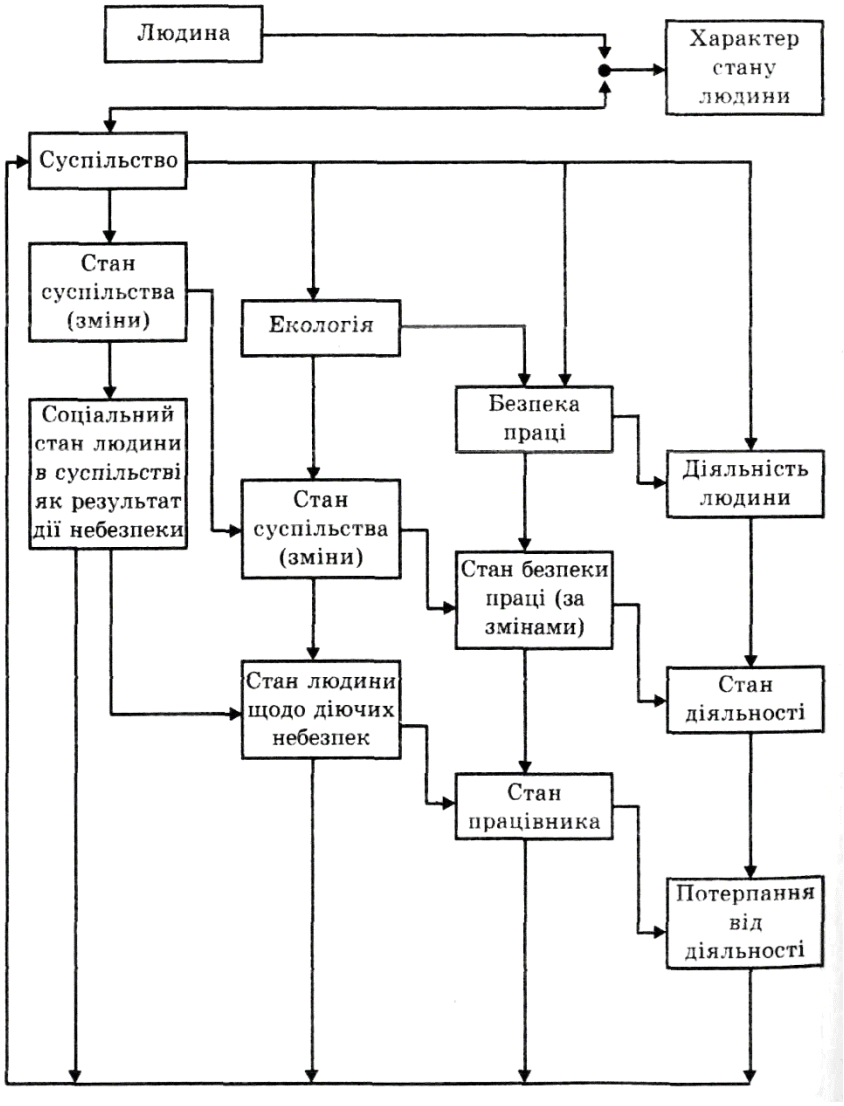


Рисунок 6.4 – Характер впливу складових життєдіяльності на стан людини

В умовах нестабільності суспільства інформація відіграє важливу роль у підтриманні злагоди в людських відносинах, а також в суспільстві. Значна частина інформації, що пов'язана з безпекою життєдіяльності за своєю природою, має негативний характер. Тому треба ретельно ставитися до змісту інформації і, головне, виключити можливість потрапляння в ЗМІ дезінформації, неперевірених фактів, перебільшених статистичних матеріалів та ін. Все це може призвести до ускладнення відносин, поглиблення конфліктів і врешті-решт до небажаних негативних наслідків. У глобальному плані дезінформація сама по собі стає джерелом небезпек, які в суспільстві в багатьох випадках бувають тяжкими.

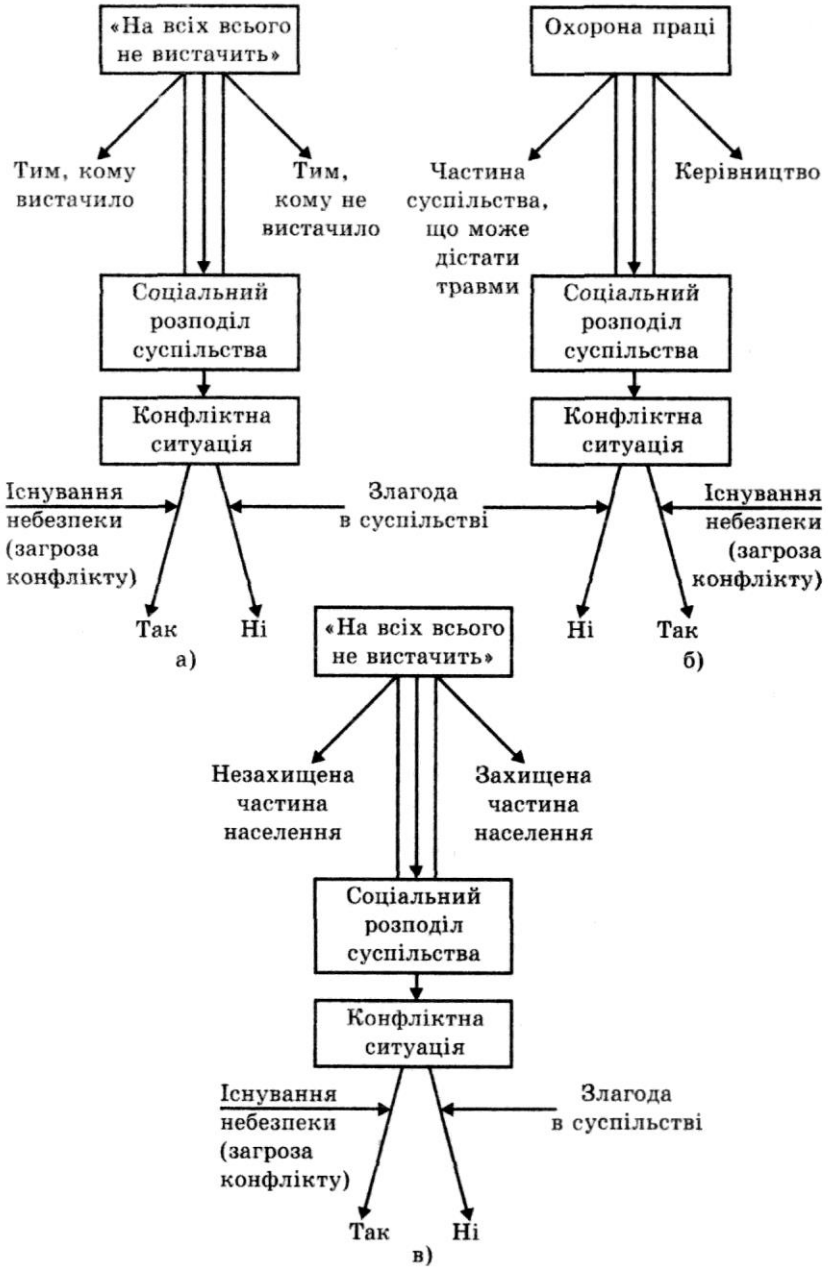


Рисунок 6.5 – Можливий розподіл суспільства залежно від ситуацій щодо стану навколишнього середовища (а), охорони праці (б), надзвичайних ситуацій (в)

Як показали останні міжнародні конфлікти в Югославії, а також в Чечні, одночасно з будь-якими конфліктами йдуть «інформаційні війни». І перемога в конфлікті ще не означає перемогу в «інформаційній війні», що автоматично перед світовим суспільством, в його уяві формує чуття глибокої кризи і моральної поразки.

Значні конфлікти в суспільстві з одночасним виникненням небезпеки пов'язуються з економічною діяльністю, особливо з її нестабільністю. Нестабільність такої діяльності в будь-який час розшаровує суспільства в умовах

падіння виробництва, зниження запитів ринку на продукцію, краху економічної системи. Тому до небезпеки слід віднести і розшарування суспільства в умовах кризи економічної діяльності.

6.2 Вимоги законодавства з охорони праці в галузі інформаційних технологій

Правову основу охорони праці становить Конституція України як за своїми юридичними особливостями, так і своїми принципами, тобто юридично вираженими об'єктивними закономірностями організації і функції соціально-економічної, політичної, духовної сфер суспільства, правового положення особи.

Конституційні норми, з одного боку, закладають суть безпеки (норми-принципи), а з іншого, – вказують на цілі подальшого розвитку і реалізацію правового забезпечення безпеки життєдіяльності (норми-програми, норми-завдання, норми-зобов'язання).

Реалізація і розвиток основних конституційних положень, які регламентують суспільні правовідносини, безпосередніми суб'єктами яких є особа і держава, здійснюється за допомогою як чинних фундаментальних нормативно-правових актів, так і спеціальних (Кодексів України про працю, Закону "Про охорону навколишнього природного середовища" та ін.)

Поруч з нормативними актами, які прийняті вищим законодавчим органом держави, для встановлення взаємозв'язків, усунення програм, а в ряді випадків і реалізації окремих правових норм або їх елементів, до правової бази безпеки життєдіяльності належать спеціальні акти, розроблені за дорученням виконавчих державних органів усіх рівнів (Кабінет Міністрів, Міністерства, Державні Комітети та ін.).

Так, наприклад, "Положення...", які розвивають Закон України Про охорону праці, діляться на звичайні "Положення" і "Типові положення". Тут держава розподілила питання своєї прерогативи стосовно розробки нормативних актів і прерогативи своїх повноважень стосовно контролю, "правового простору" у вигляді нормативних актів підприємств.

З іншого боку, формуючи систему "Типових положень" держава на сьогоднішній день ліквідує прогалини в чинному законодавстві, узгоджує взаємозв'язки між суб'єктами правовідносин, створює юридичну базу для удосконалення і розвинення "правового поля" підприємств.

Кожний нормативно-правовий документ має свою структуру, яка визначає собою ідею систематизації відповідно зі своїм рівнем, метою та завданнями. Відповідно до цього в кожному нормативному акті є елементи, що відповідальні за зовнішній його зв'язок і створення передумов для відповідного розвинення за рахунок розробки нижчих нормативно-законодавчих актів. Сама структура нормативного акта формує відповідні внутрішні зв'язки.

Основними систематизуючими ланками нормативних актів безпеки життєдіяльності (які за ієрархією знаходяться нижче законів) є встановлення взаємовідносин в галузі виробництва, в межах дії небезпечного фактора (в тому числі і факторів довкілля), а також відносно управління основних технологій безпеки життєдіяльності (розслідування нещасних випадків, навчання, організації робіт та ін.).

Узагальнюючими ланками систематизації на рівні держави є національна ідея, взаємовідносини в суспільстві, соціально-економічне і політичне становище держави, можливості сприймання і використання законодавчих актів з боку споживачів та ін.

6.3 Розрахунок освітленості робочого місця розробника модуля інформаційної системи для збору статистики про відвідування сайту

Належне освітлення необхідне для виконання більшості задач програміста, а, отже, і розробника даного дипломного проекту. Для того, щоб спланувати раціональну систему освітлення, враховується специфіка робочого завдання, для якого створюється система освітлення, швидкість і точність, з якою це робоче завдання повинне виконуватися, тривалість його виконання і різні зміни в умовах виконання робітників.

Приміщення, у якому знаходиться робоче місце, має наступні характеристики:

- довжина приміщення 16 м;
- ширина приміщення 6 м;
- висота 4 м;
- кількість вікон 3;
- кількість робочих місць 3;
- біла стеля, блідо-зелені стіни, підлога обтягнута лінолеумом зеленого кольору.

У приміщенні, де знаходиться робоче місце, використовується змішане освітлення, тобто сполучення природного і штучного освітлення.

У якості природного – бічне освітлення через вікна. Штучне освітлення використовується при недостатньому природному освітленні.

Розрахунок його здійснюється по методу світлового потоку з врахуванням потоку, відбитого від стін і стелі.

Нормами для даних робіт встановлена необхідна освітленість робочого місця Ен = 300 лк [17]. Загальний світловий потік визначається за формулою:

$$F_{заг} = \frac{E_н * S * z1 * z2}{V}, \quad (3.1)$$

де $E_н$ – нормована освітленість ($E_н$ = 300 лк);

S – площа приміщення;

z1 – коефіцієнт, що враховує старіння ламп і забруднення світильників (z1 = 1,5) [17];

z2 – коефіцієнт, що враховує нерівномірність освітлення приміщення (z2 = 1,1) [17];

V – коефіцієнт використання світлового потоку; визначається в залежності від коефіцієнтів відбивання від стін, стелі, робочих поверхонь, типів світильників і геометрії приміщення.

Площа приміщення S = A*B = 16*6 = 96 м^2.

- коефіцієнт відбивання побіленої стелі $R_п$ = 70% [17];
- коефіцієнт відбивання від стін, пофарбованих у світлий колір $R_{ст}$ = 50% [17];
- коефіцієнт відбивання від підлоги, покритого лінолеумом темного кольору $R_р$ = 10% [17];
- індекс приміщення $i = \frac{A*B}{h*(A+B)} = \frac{16*6}{4*(16+6)} = 1,1$.

Знайдений коефіцієнт V = 0,34.

За формулою (3.1) визначаємо загальний світловий потік

$$F_{заг} = \frac{300*96*1,1*1,5}{0,34} = 139764 \text{ лм.}$$

Для організації загального штучного освітлення виберемо лампи типу ЛБ40. Люмінесцентні лампи мають ряд переваг перед лампами накалювання: їхній спектр ближче до природного; вони мають велику економічність (більша світловіддача) і термін служби у 10-12 раз більший.

Поряд з цим вони мають і недоліки: їхня робота супроводжується іноді шумом; гірше працюють при низьких температурах; не можна використовувати у вибухонебезпечних приміщеннях.

Для нашого приміщення люмінесцентні лампи підходять. Світловий потік однієї лампи ЛБ40 складає не менш $F_л$ = 2810 лм. Число N ламп, необхідних для організації загального освітлення визначається наступним чином:

$$N = \frac{F_{заг}}{F_л} = \frac{139764}{2810} = 50 \text{ шт.}$$

Як світильники вибираємо ПВЛ-1, 2*40 Вт. Таким чином, щоб забезпечити світловий потік $F_{заг}$ = 139764 лм треба використати 25 світильників по 2 лампи ЛБ40 у кожному.

Електрична потужність однієї лампи $W_л$ = 40 Вт. Потужність всієї освітлювальної системи:

$$W_{заг} = W_л * N = 40*50 = 2000 \text{ Вт.}$$

Зі зробленого в даному розділі розрахунку випливає, що для нормальної роботи користувача робочого місця необхідне загальне освітлення приміщення зі світловим потоком 139764 лм, для чого необхідна наявність 25 світильників типу ПВЛ-1 з двома лампами типу ЛБ40. Крім того рекомендується використовувати ряд спеціальних заходів захисту від шкідливих факторів екрана дисплея, наприклад, використання занавісок на вікнах.

6.4 Аналіз небезпеки і шкідливості при розробці програмного забезпечення для аналітичного аналізу текстових даних

Організація робочого місця розробника модуля впливає на його працездатність.

У своїй діяльності розробник використосвує комп’ютер, пристрої збереження інформації, а тому є необхідність забезпечення зручного доступу до всіх технічних засобів. Тому в даному розділі докладніше розглянемо відомості про систему ергономічних норм і принципів організації робочого місця, на котрому проводяться роботи зі створення модуля збору статистики.

Під робочим місцем розуміється зона, оснащена необхідними технічними засобами, у якій відбувається трудова діяльність виконавця або групи виконавців, які спільно виконують одну роботу або операцію.

Організація робочого місця полягає у виконанні заходів, які забезпечують безпечний і раціональний трудовий процес і ефективне використання знарядь та предметів праці, що підвищує продуктивність праці і знижує стомлюваність працівника.

Організація робочого місця залежить від характеру розв'язуваних задач і особливостей предметно-просторового оточення, що визначають робоче

положення тіла і можливість пауз для відпочинку, типи і способи засобів відображення і керування, необхідність у засобах захисту, спецодягу, простору для налагодження і ремонту устаткування.

Одним з компонентів діяльності на робочому місці є робочі рухи. Їхня раціональна організація створює умови для зниження стомлення, резерви для підвищеної працездатності. Просторові характеристики руху оператора визначаються траєкторіями руху і розмірами моторного поля (зони досяжності).

При організації робочого місця необхідно забезпечити нормальні умови огляду. Зону огляду описує кут, вершина якого знаходиться в центрі ока, а сторони складають границі, в яких людина при фіксованому положенні голови й ока добре розрізняє їхнє місцезнаходження.

У горизонтальній площині цей кут складає 30^0 – 40^0. При організації робочого місця кут огляду можна взяти 50^0 – 60^0, включаючи зону менш ясного огляду. Допустимий кут огляду по горизонталі 90^0. У вертикальній площині оптимальний кут огляду 10^0 вверх і 30^0 вниз від лінії погляду, а допустимий 30^0 вгору і 40^0 вниз від лінії погляду.

Щоб зберегти нормальну гостроту зору, робочу поверхню розташовують від очей на відстані від 0,3 м до 0,75 м. Робочі меблі повинні бути зручними для виконання робочих операцій. В даному випадку робочий стіл є основним устаткуванням. Особливо важливе значення має висота столу, його конструкція, яка повинна передбачати шухляди для розміщення інструментів, документації.

Важливе значення має конструкція робочих крісел. Погано підібрані крісла можуть бути причиною надмірної стомлюваності.

Нахил і висота крісла повинні регулюватися відповідно до висоти робочої поверхні і росту працюючого. Рекомендована ширина крісла 370 – 400 мм, глибина 370 – 420 мм, висота спинки 370 – 1000 мм від рівня крісла. Для розміщення ніг необхідно передбачити вільний простір під робочою площиною [17].

Праця людини, що протікає в умовах надмірного нервово-емоційного напруження, довготривалих статичних навантажень, обмеженої рухової активності призводить до неврозів, відхилень у психіці, захворювань опорно-

рухового апарату, серцево-судинної системи тощо. Комп'ютери, телебачення, системи зв'язку та інші засоби, що використовують досягнення радіоелектроніки, є генераторами цілої низки електромагнітних випромінювань, вплив яких на організм людини ще не зовсім вивчений.

Сучасний розвиток науки та техніки приносить принципові нововведення у всі сфери матеріального виробництва, докорінно змінюючи знаряддя та предмети праці, технологію, методи обробки інформації. Разом з тим, захопившись вдосконаленням засобів праці залишено поза увагою проблеми людини в рамках своєрідної технічної та комп'ютерної революції. З широким впровадженням автоматизації та комп'ютеризації виникла потреба врахування психологічних можливостей людини, таких як швидкість реакції, особливості пам'яті та уваги, емоційний стан та ін. Поява операторської діяльності призвела до суттєвих змін у фаховій структурі праці. Зменшились фізична важкість праці, ризик виробничого травматизму, однак разом з тим, на працюючу людину посилюється вплив нових, раніше не відомих чи мало вивчених несприятливих виробничих факторів фізичного, хімічного і особливо психофізіологічного характеру.

Проте, розвиток сучасної обчислювальної техніки відбувається не лише у бік поктащення її технічних параметрів, але також звертається увага безпеку використання цієї техніки людиною шляхом зменшення потужності випромінювачів, зменшенням рівня випромінювання з моніторів, зменшення напруг живлення, покращення ергономічних характеристик.

Таким чином, в розділі з охорони праці виконано огляд питань безпечної роботи при створенні модуля інформаційної системи збору статистики та встановлено, що умови такої роботи відповідають вимогам з охорони праці, які застосовуються в галузі інформаційних технологій.

7 ЕКОЛОГІЯ

7.1 Програмне забезпечення еколого - статистичних досліджень

Оперативна, якісна і точна обробка великих масивів статистичної інформації може бути виконана лише з використанням сучасних засобів обчислювальної техніки. Наявність потужних, надійних і разом з тим простих в експлуатації програмних продуктів статистичного аналізу звільняє дослідника від рутинних операцій, розширює сферу застосування статистичних методів в різних галузях людської діяльності, сприяє появі якісно нових можливостей статистичного аналізу і моделювання даних. Використання пакетів прикладних програм – це єдиний реальний практичний інструмент розв'язування задач багатофакторного кореляційно-регресійного аналізу в багатовимірному просторі.

Програмне забезпечення статистичних досліджень досить розвинуте. Сучасний ринок програмних продуктів пропонує різноманітні пакети програм для статистичної обробки даних. Всесвітньо відомі статистичні пакети для комплексної обробки даних: BMDP, SPSS, SAS, Systat, Minitab, S-Plus, Statgraphics Statistica та інші.

Використання згаданих пакетів програм дає змогу автоматизувати процес статистичного дослідження в таких напрямах:

- створення файлів даних і таблиць;
- групування даних;
- графічний аналіз даних;
- розрахунок варіаційних характеристик вибіркових сукупностей;
- побудова рядів розподілу;

– аналіз рядів динаміки і прогнозування їх майбутніх рівнів;
– кореляційно-регресійний аналіз;
– багатомірний аналіз.

З 1995 р. Світовим лідером на ринку статистичного програмного забезпечення визнається інтегрована система Statistica для Windows (версія 7.0), розроблена фірмою Stat Soft. Перша версія програми з'явилася у 1991р. для операційної системи MS-DOS і була новим напрямом розвитку статистичного програмного забезпечення. В ній реалізовано графічно-орієнтований підхід до статистичного аналізу даних, суть якого полягає в отриманні всебічного візуального представлення інформації на всіх етапах статистичної обробки даних.

Багатофункціональна, графічно орієнтована на обробку масових даних система Statistica відповідає основним стандартам Windows (динамічний обмін даними з іншими додатками, підтримка основних операцій з буфером обміну, робота в мережевому середовищі та інші).

7.2 Вимоги до моніторів (ВДТ) і ПЕОМ

Монітор, як і будь-який пристрій має відповідати певним вимогам, і стандартам. Вимоги на монітори поділяють на дві основні групи стандартів і рекомендацій - про безпеку і ергономіку.

До першої групи ставляться стандарти UL, CSA, DHHS, CE, скандинавські SEMRO, DEMKO, NEMKO, FCC Class B. З другої групи найвідоміші MPR-II, TCO’92, TCO’95, ISO 9241-3, EPA Energy Star, TUV Ergonomie.

1. FCC Class B - цей стандарт розроблений канадською федеральною комісією з комунікацій задля забезпечення прийнятного захисту довкілля. Устаткування, яке відповідає вимогам FCC Class B, не повинно заважати роботі теле- і радіо апаратурі.

2. MPR-II - цей стандарт був випущений у 1990 р. Шведським національним департаментом. MPR-II накладає обмеження на випромінювання від комп'ютерних моніторів і промислової техніки, яка в офісі.

3. TCO’92 (TCO’95) - рекомендація, розроблена Шведською конференцією профспілок та Національною радою індустріального і технічного розвитку Швеції (NUTEK). Регламентує взаємодію Космосу з навколишнім середовищем. Вона потребує зменшення електричного і магнітного полів до технічно можливого рівня з метою захисту користувача. Щоб отримати сертифікат TCO’92, монітор повинен відповідати стандартам низького випромінювання (Low Radiation), тобто мати низький рівень електромагнітного поля.

4. TUV Ergonomie - німецький стандарт ергономіки. Монітори, що відповідають цьому стандарту, пройшли випробування відповідно до EN 60950 (електрична безпека) і ZN 1/618 (ергономічне облаштування робочих місць, оснащених дисплеями), і навіть відповідають шведському стандарту MPR-II.

5.EPA Energy Star VESA DPMS - відповідно до цього стандарту монітор повинен підтримувати три енергозберігаючих режиму - очікування (stand-by), припинення (suspend) і “сон” (off). У режимі чекання зображення на екрані пропадає, але внутрішні компоненти монітора функціонують у нормальному режимі, а енергоспоживання знижується до 80%. У режимі припинення, зазвичай, відключаються високовольтні вузли, а споживання падає до 30 Вт і менше. Та у режимі “сну” монітор споживає трохи більше 8 Вт, а функціонує в нього лише мікропроцесор. При натисканні будь-якої клавіші клавіатури чи русі миші монітор перетворюється на нормальний режим роботи.

6. Російський стандарт ГОСТ 27954 – 88. Даним стандартом регламентується ступінь деталізації технічної документації на моніторі, встановлюються вимоги стандартизації, та уніфікації, технологічності, ергономіки та програмах технічної естетики, екологічну безпеку, технічного ремонту й обслуговування, і навіть надійності.

Монітори персональних комп'ютерів, і робочих станцій при обов'язкових сертифікаційних випробуваннях мають такі параметри:

1. Параметри безпеки - електрична, механічна, пожежна безпеку (ГОСТ Р 50377 - 92).

2. Санітарно - гігієнічні вимоги - рівень звукових шумів (ГОСТ 26329 - 84 чи ГОСТ 2718 - 88), ультрафіолетове, рентгенівське випромінювання та показники якості зображення (ГОСТ 27954 - 88).

3. Электромагнитная сумісність - випромінювані радіоперешкоди (ГОСТ 29216 - 91).

Сертифікат видається лише при відповідності усім переліченим вище ГОСТам.

Також рекомендується наявність екранами моніторів антистатичного покриття (antistatic coating) - яке перешкоджає виникненню на поверхні екрана електростатичного заряду, що притягує пилюку і не сприятливо впливає для здоров'я користувача.

Конструкція монітора (відео-дисплейного терміналу – ВДТ) повинна забезпечувати можливість фронтального спостереження екрана шляхом повороту корпуса в горизонтальній площині навколо вертикальної осі в межах ± 30 ° і у вертикальній площині навколо горизонтальної осі в межах ± 30 ° з фіксацією в заданому положенні. Дизайн моніторів повинен бути в спокійних м'яких тонах з дифузним розсіюванням світла. Корпус монітора і ПЕОМ, клавіатура повинні мати матову поверхню одного кольору з коефіцієнтом відображення 0,4 – 0,6 і не мати блискучих деталей, здатних створювати відблиски.

Конструкція ВДТ повинна передбачати наявність ручок регулювання яскравості і контрасту, що забезпечують можливість регулювання цих параметрів від мінімальних до максимальних значень.

ВДТ і ПЕОМ мають забезпечувати потужність експозиційної дози рентгенівського випромінювання у будь-якій точці з відривом 0,05м від екрану й корпусу монітора.

Приміщення з моніторами ПЕОМ та ВДК повинні мати природне і штучне освітлення. Природне освітлення має здійснюватися через вікна, орієнтовані переважно на північ і північний схід, забезпечувати коефіцієнт

природного освітлення (КЕО) не нижче 1,2 % в зонах зі стійким сніжним покровом і нижче 1,5 % на іншій території. Зазначені значення КЕО нормуються для будинків, розміщених у III світловому кліматичному поясі.

У диспетчерських, операторських, розрахункових кабінах, залах обчислювальної техніки та інших з моніторами, де працюють інженерно - технічні працівники, здійснюють лабораторний, аналітичний чи вимірювальний контроль, рівень шуму не повинен перевищувати 60 дБА.

У приміщеннях операторів ЕОМ (без дисплеїв) рівень шуму не повинен перевищувати 65 дБА.

На робочих місцях помешкань рівень шуму з устаткуванням таким, як, наприклад, АЦДП (алфавітно-цифровий друкуючий пристрій), принтери та інші, не має перевищувати 75 дБА.

Шумливе устаткування (АЦДП , принтери та інші), рівні шуму якого перевищують нормовані, має перебуває поза приміщенням з монітором і ПЕОМ.

Знизити рівень шуму, зчиненого на помешкання із моніторами і ПЕОМ можна використанням звукопоглинальних матеріалів з максимальними коефіцієнтами звукопоглинання у діапазоні частот 63 - 8000 Гц для обробки приміщень, підтверджених спеціальними акустичними розрахунками.

Додатковим звукопоглинанням служать однотонні фіранки з щільної тканини. Ширина фіранки має бути у 2 рази більше ширини вікна.

Схеми розміщення робочих місць із ВДТ і ПЕОМ повинні враховувати відстані між робочими столами з відео моніторами (у бік тилу поверхні одного відео монітора і екрана іншого відео монітора), яке повинно мати не мене 2,0 м, а відстань між бічними поверхнями відео моніторів - щонайменше 1,2 м.

Згідно з ДНАОП 0.00-1.81-99 в Україні забороняється використання нових несертифікованих ВДТ. Сертифікат має бути виданий державною системою сертифікації УкрСЕПРО, що засвідчує їх відповідність обов'язковим вимогам.

ВДТ, що перебувають в експлуатації на час уведення в дію ДНАОП 0.00-1.31-99 і не мають відповідного сертифіката, мають пройти експертизу їх безпечності та нешкідливості для здоров'я людини, відповідності до вимог

чинних в Україні нормативних документів в установах, які мають дозвіл органів Державного нагляду за охороною праці на проведення такої роботи.

ВИСНОВКИ

Існує велика кількість різноманітних патернів і архітектур програмування для рішення поставлених задач і для більш комфортної роботи їх краще каталогізувати.

Розроблена технологія дає можливість запровадити поетапний процес підтримки прийняття експертних рішень по архітектурі програмних додатків різних типів при достатньо великій кількості альтернатив для кожного типу, з можливістю вибору найкращої архітектури.

Розроблена програма є актуальною, оскільки надає можливість на основі створеного репозиторію будувати архітектури різних типів програмних додатків і виконувати їх порівняльний аналіз та оцінювання з метою визначення найбільш ефективної архітектури с точки зору встановлених властивостей якості та критеріїв для визначених типів програмних додатків.

Підсумовуючи виконану роботу, можемо стверджувати, що результатом проектування та реалізації є програмний комплекс створення репозиторію патернів функціональних модулів та каркасів програмних архітектур, в складі якого містяться шари, функціональні модулі та різноманітні патерни для представлення програмних додатків. Результат демонструє можливості біль швидкого огляду і оцінки різних архітектурних рішень.

ПЕРЕЛІК ПОСИЛАНЬ

1. І.О. Боднарчук Експертна система проектування архітектури програмного забезпечення : О.Г. Харчеко, І.О. Боднарчук, В.В. Яцишин, 2013 р.

2. Звіт про науково-дослідну роботу №876 ДБ 13 розробка, дослідження, та впровадження методів і засобів контролю та управління якістю програмних продуктів : УДК 004.415.5 КП : Харченко О.Г., Райчев І.Е. Щербак О.А., Павленко Б.С.

3. Харченко О. Г. Експерта система проектування архітектури програмного забезпечення / О. Г. Харченко, І. О. Боднарчук, В. В. Яцишин // Комп'ютерні технології друкарства. – № 29. – 2013. – С. 10-26.

4. Брауде Е. Технология разработки программного обеспечения / Е. Брауде – СПб. : Изд-во "Питер", 2004. – 655 с.

5. Метод багатокритеріальної оптимізації програмної архітектури на основі аналізу компромісів [Текст] / Харченко О.Г., Боднарчук І.О., Галай І.О. // Інженерія програмного забезпечення. – 2012. – № 3–4 (11–12). – с. 5–11.

6. Гамма, Э. Приемы объектно-ориентированного проектирования. Паттерны проектирования [Текст] / Э. Гамма, Р. Хелм, Р. Джонсон, Д. Влиссидес. – СПб. : Питер, 2010. – 366 с. – ISBN 978-5-469-01136-1.

7. Руководство Microsoft по проектированию архитектуры приложений. 2-е издание. Microsoft, 2009. – 529 с

8. Буч Г. UML: специальный справочник / Г. Буч, Дж. Рамбо, А. Джекобсон – СПб.: Питер, 2002.– 656 с.

9. ДСТУ ISO 9001 – 2001. Системи управління якістю. Вимоги. – Чинний від 27.06.2001. – К. Держстандарт України, 2001 – 23 с.

10. Саати Т. Принятие решений. Метод анализа иерархий / Т. Саати – М.: Радио и свіязь – 1993 – 315 с.

11. Фаулер М. Архитектура корпоративных программных приложений /М. Фаулер – Пер. с англ. – М.: Издательский дом "Вильямс" – 2006. – 544 с.

12. ISO/IEC 9126-1. Software engineering – Product quality – Part 1: Quality model, 2001 – 26 p.

13. ISO/IEC TR 9126-2. Software engineering – Product quality –Part 2: External metrics, 2003 – 86 p.

14. ISO/IEC TR 9126-3. Software engineering – Product quality – Part 3: Internal metrics, 2003 – 66 p.

15. ISO/IEC TR 9126-4. Software engineering – Product quality – Part 4: Quality in use metrics, 2004 – 70 p.

16. Методичні вказівки по виконанню організаційно-економічної частини дипломних проектів науково-дослідницького характеру для студентів спеціальності 7.080401 “Інформаційні управляючі системи та технології” / Кирич Н.Б., Зяйлик М.Ф., Брощак І.І., Шевчук Я.М – Тернопіль, ТНТУ, – 2009. –11 с.

17. Основы охраны труда: учебник / А. С. Касьян, А. И. Касьян, С. П. Дмитрюк. – Дн-ськ: Журфонд, 2007. – 494 с.

18. Безпека життєдіяльності: Навч. посібник./ За ред. В.Г. Цапка. 4–те вид., перероб. і доп. – К.: Знання, 2006. – 397 с.