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: **Проект програмної системи оптимізації архітектурних рішень в процесі проектування програмного забезпечення**

Виконав: студент 6 курсу, групи СТм-61
спеціальності 126 «Інформаційні системи та технолгії»
(шифр і назва спеціальності)

_____ Метохір С.В.
(підпис) (прізвище та ініціали)

Керівник _____ Струтинська І.В.
(підпис) (прізвище та ініціали)

Нормоконтроль _____ Мацюк О.В.
(підпис) (прізвище та ініціали)

Рецензент _____ _____
(підпис) (прізвище та ініціали)

м. Тернопіль – 2019

АНОТАЦІЯ

Проект програмної системи оптимізації архітектурних рішень в процесі проектування програмного забезпечення // Дипломна робота ОР "Магістр" // Метохір Соломія Володимирівна // 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, ПРОГРАМНИЙ МОДУЛЬ, ЯКІСТЬ ПРОГРАМНИХ СИСТЕМ, МОДЕЛІ ЯКОСТІ, РОЗРОБКА ВИМОГ, МОДИФІКАЦІЯ ЖИТТЄВОГО ЦИКЛУ, ОЦІНЮВАННЯ ЯКОСТІ ПРОГРАМИХ СИСТЕМ, ІНФОРМАЦІЙНА СИСТЕМА.

Основна частина дипломної роботи складається з трьох розділів. У першому розділі розглянуті теоретичні відомості про життєвий цикл ІС, методологію розробки каркасів архітектур, описані головні складові частини програмних архітектур, розглянуті методи оцінювання альтернатив.

Другий розділ описує загальну концепцію роботи розробленого програмного комплексу, розкриває функціональну структуру системи, описує використаний метод оцінювання альтернатив. Містить в собі UML діаграми, що описують систему.

У третьому розділі представлена реалізація програмного комплексу, методологія його використання та описане середовище розробки даного програмного комплексу.

Метою проектування є розгляд теоретичних засад створення багатошарової архітектури, формалізація процесу проектування архітектур, розробити математичну модель оцінювання якості альтернативних архітектур, розробити алгоритм процесу прийняття рішень на основі

оцінювання, розробити проект програмного комплексу, що реалізує поставлені задачі.

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

В роботі запропоновані варіанти компонування альтернативних архітектур, розроблений механізм формування альтернатив їх попарного оцінювання та виділення найбільш оптимальної.

Розроблений проект програмного комплексу показує можливості раннього оцінювання архітектурних рішень на етапі проектування системи.

Можливі напрямки розвитку роботи пов’язані з розширенням можливості варіативних архітектур, додавання інших методів оцінювання та розширенням загальної функціональності.

Дипломна робота пройшла апробацію на студентській конференції у вигляді публікації тез доповіді.

ANNOTATION

The project of the software system for architecture decisions optimization during software design // Diploma paper of Master degree level // Metokhir Solomiya Volodymyrivna // 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, DESIGN PATTERNS, SOFTWARE UNIT, SOFTWARE SYSTEMS QUALITY, QULITY MODELS, REQUIREMENTS AQUISIONING, LIFE CYCLE MODIFICATION, SJFTWARE SYSTEMS QUALITY ASSESSMNET, INFORMATION SYSTEM.

The main part of the diploma paper consists of three chapters. First one contains theoretical background of information systems life cycle, the technics of software architectures design, the methods of alternatives assessments are discussed.

The second chapter describes general concepts of developed software operating, disclosure functional structure of the system, describes used method for assessment of alternatives. Contains within UML charts that describe the system.

The implementation of software complex is presented in the third chapter. The methods of its using and integrated development environment is described.

The goal of the diploma paper is to discuss theoretical background for multitier architecture design, formalization of software architecture design, to develop mathematical model of alternative architectures quality evaluation, to develop the algorithm of decision making on the base of evaluation, to develop the design of software that implements stated problems.

The subject of research is a software? Theoretical background of alternative architectures composing and software complex that forms the set of alternative architectures and following their comparative assessment and general choosing of most optimal architecture on the base of expert assessment.

The variants of composing of alternative architectures are offered in the paper, the method of forming of alternative architectures and their comparison with highlighting of most optimal is offered as well.

The diploma paper was approbated on students' conference as a thesis.

Developed software complex demonstrates possibilities of for variative architectures, adding another methods of evaluation and expanding of general functionality.

The diploma paper is approbated on students' conference as a thesis

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. Тому актуальним є застосування формальних методів, таких як математичне моделювання, оптимізація, теорія прийняття рішень для розробки моделей процесів проектування та побудови на їх основі засобів автоматизації підтримки прийняття рішень проектувальниками.

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 узагальній кооперативній діяльності, для якої визначені ролі учасників і їхня відповідальність.

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. Найбільш відомими з них є методи, які базуються на розробці сценаріїв використання та перевірки, чи задовольняє даний варіант архітектури вимозі по певному критерію якості.

Актуальність теми дипломної роботи виражається в тому, що вона описує методику створення множини альтернативних архітектур для вирішення однієї задачі на етапі проектування . Оцінювання цих архітектур та виділення серед них найоптимальніших по даному критерію якості, чи по комплексному. Для подальшого використання даного каркасу архітектури для розробки безпосередньої системи.

Об'єкт дослідження – процес проектування архітектури програмного забезпечення.

Предметом дослідження є цикл розробки архітектурних рішень для вирішення поставленої задачі перед архітектором. Також оцінювання створених альтернативних архітектур для використання каркасу в розробці.

Метою даної дипломної роботи є розгляд теоретичних та практичних засад технології побудови архітектури програмних додатків, формалізацію створення архітектур та оцінки їх.

Для дослідження цілі дипломної роботи поставлені наступні задачі:

- розглянути основні види архітектур програмних систем;
- розглянути методи оцінки архітектурних рішень та критерії оцінювання;
- розробити програмний комплекс по створенню альтернатив та оцінюванню їх;
- задокументувати розроблений програмний комплекс.

1 АКТУАЛЬНІСТЬ ТА ОБГРУНТУВАННЯ ПРОБЛЕМИ

1.1 Постановка задачі автоматизації етапів життєвого циклу інформаційної системи

Сучасний стан розвитку ІС характеризується постійним зростанням їхньої складності. Це потребує залучення значних трудових і часових ресурсів для забезпечення ефективності їх функціонування. Хоч сьогодні розроблено цілу низку методів та методологій проектування ІС, як корпоративних, так і загального використання, але при їх застосуванні часто виникають проблеми, пов'язані з недостатньою формалізацією процесів проектування. А це в свою чергу перешкоджає впровадженню засобів автоматизації цих процесів.

Технологія створення ПС базується на процесах ЖЦ, які за своєю природою є досить складними, трудомісткими і творчими. У зв'язку з цим виникає потреба створення CASE-технологій, які б надавали змогу автоматизувати операції та стадії як основних, так і допоміжних процесів ЖЦ. Розроблення таких технологій є актуальною задачею, оскільки їх застосування підвищує ефективність процесу розроблення програмних продуктів приблизно у 6 разів.

Аналіз процесів ЖЦ розроблення ПС показав, що найменш формалізованими і тому найбільш трудомісткими є процеси на етапі аналізу вимог та оцінювання якості ПС. Це пояснюється складністю використання формальних методів для відображення потреб замовника на специфікації системних вимог, що породжується відсутністю єдиного уніфікованого їх подання.

Чинний стандарт IEEE 830 дає достатньо довільні трактування структури та форми представлення вимог, що відображається у неоднозначності та неузгодженості у сформульованих вимогах. Звідси

випливає, що для створення ефективних CASE-засобів автоматизації процесів розроблення специфікацій вимог, їх контролю та управління необхідно використовувати стандартизовані уніфіковані моделі їх представлення і відповідні процедури для їх побудови.

Оскільки моделі вимог якості, які базуються на уніфікованих компонентах і містять критерії оцінювання, є основою для процедур оцінювання якості, то вони також можуть бути базою для створення CASE-засобів оцінювання якості ПС.

До основних технологічних процесів, які вимагають автоматизації на етапі розробки вимог до ПС та при застосуванні запропонованого підходу моделей якості, належать наступні [4]:

1. Процес збору та документування потреб замовника у ПС.
2. Процеси формування і збереження стандартизованих критеріїв якості.
3. Процес формалізації та документування атрибутів предметної області.
4. Процес відображення потреб замовника у вимоги, які подаються в уніфікованих термінах моделей якості.
5. Процес забезпечення контролю і керування вимогами.
6. Процес специфікації вимог.

В загальному випадку функціональну залежність процесів, які необхідно автоматизувати на етапі розробки вимог, можна зобразити у вигляді схеми, яка зображена на рисунку 1.1.

Необхідність автоматизації процесів збору потреб та визначення атрибутів предметної області пов’язана з великим обсягом даних та їх структурною складністю. У зв’язку з цим автоматизація процесу збору та документування потреб вимагає організації репозиторію для збереження та керування ними, а також розробки відповідного користувацького інтерфейсу.

Це також стосується автоматизації процесу аналізу та документування атрибутів предметного середовища.

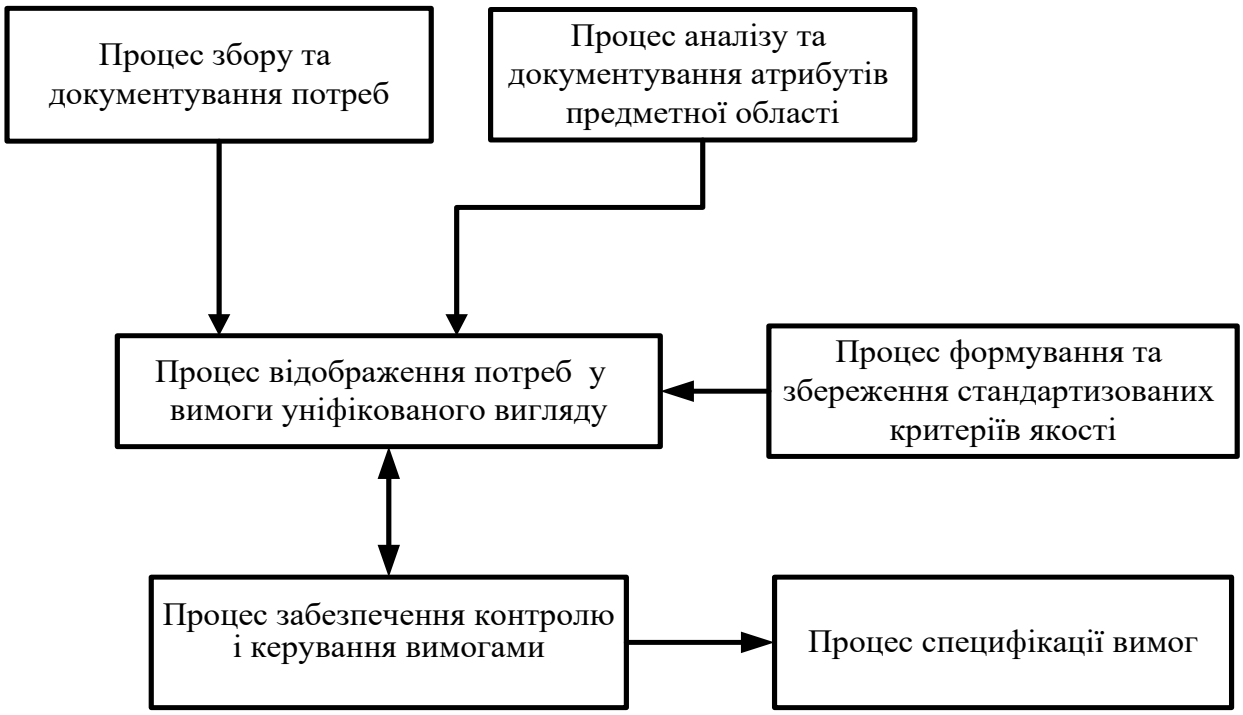


Рисунок 1.1 – Зв’язок процесів, які необхідно автоматизувати на етапі розробки вимог до ПС

Важливим з погляду автоматизації є розробка репозиторію для зберігання та керування стандартизованими критеріями якості, оскільки лише до складу зовнішньої моделі якості входить 6 основних характеристик та 27 підхарактеристик [12], а крім цього кожна з підхарактеристик має свій набір атрибутів та відповідних їм метрик. Це породжує потужну множину даних, які доволі складно зберігати не автоматизованим способом.

Процес відображення потреб у вимоги стандартизованого вигляду передбачає формалізацію атрибутів предметного середовища та тих, які визначені на основі потреб замовника. Крім цього, згідно запропонованого у розділі 2 підходу, процес відображення потреб у вимоги вимагає проведення класифікації атрибутів за стандартизованими характеристиками та підхарактеристиками моделей якості, а також визначення пріоритетності

вимог для забезпечення їх несуперечності і розв’язання конфліктності між ними. У зв’язку з тим, що процедури класифікації атрибутів за характеристиками моделей якості, процедури розв’язання конфліктних ситуацій та визначення пріоритетності вимог є досить трудомісткими і вимагають значних затрат часу на їх виконання, виникає необхідність комплексної автоматизації цього процесу.

Для забезпечення ефективності процесу комунікації вимог, контролю їх зміни, додавання та видалення з репозиторію даних необхідно реалізувати інструмент, який би дозволив проводити відповідні процедури. Це пов’язано з тим, що при «ручному» відстеженні вимог ймовірність ефективного керування ними нижча в десятки разів, оскільки виконання процедур відбувається не централізовано та вимагає залучення значних трудових ресурсів, як збоку розробника ПС так і зі сторони замовника.

Автоматизація процесу специфікації вимог до ПС зумовлена наявністю великих обсягів інформації та необхідністю її структуризації. Оскільки дані в процесі специфікації вимог отримують з різних джерел та різного типу, то «ручне» виконання цієї процедури може негативно вплинути на терміни виконання проекту, а також вилитись у збільшення фінансових затрат, при цьому наявність помилок у специфікації не виключена тому, що наявним є людський фактор.

Провівши аналіз технологічних процесів на стадії розробки вимог до ПС можна зробити висновок про важливість та необхідність їх автоматизації, що обумовлено складністю проведення відповідних процедур та участю у них великої кількості представників як замовника так і розробника.

Процес оцінювання якості як готового програмного продукту, так і результатів його проектування на різних стадіях ЖЦ, вимагає залучення експертних груп, основним завданням яких є визначення міри відповідності результатів проектування висунутим критеріям якості. У зв’язку з цим необхідно автоматизувати роботу експертів, забезпечивши їх відповідними

діалоговими інтерфейсами та надати можливість автоматизованого проведення оцінки згідно процедур.

Для визначення технологічних процесів, які необхідно автоматизувати, проведемо аналіз процедур процесу проектування оцінювання якості ПС, процесу його реалізації та процедури до підготовки проведення процесу оцінювання якості ПС.

Вхідними даними для процесу проектування при оцінюванні якості ПС є специфікація вимог. Оскільки, вимоги до ПС спроектовано у вигляді моделей якості, то їх специфікацію можна прямо використовувати на етапі проектування оцінювання якості. Виходячи з цього, автоматизацію слід почати перш за все з надання можливості спільного використання репозиторію вимог, а це вимагає розробки відповідного інтерфейсу для експертної групи.

Крім цього, на етапі проектування оцінювання якості виникає необхідність автоматизації процедури вибору метрик, формування елементарних функцій для оцінки атрибутів ПС та розробки критеріїв глобальних критеріїв оцінювання. Це пов’язано з тим, що сукупність операцій, які при цьому виконують експерти є досить трудомісткими та затратними по часових рамках.

В процесі реалізації оцінювання якості ПС необхідно забезпечити автоматизацію обчислення елементарного значення атрибутів, частинного та глобальних показників якості, що пов’язано з тими ж чинниками, що й в процесі проектування оцінювання якості ПС. В загальному випадку процедури, які варто автоматизувати в загальному процесі оцінювання якості представлено у вигляді таблиці 1.1.

Виходячи з результатів проведеного аналізу процесу оцінювання якості ПС можна зробити висновок про доцільність автоматизації процедур, наведених у таблиці 1.1. Це пов’язано з можливістю підвищення загальної продуктивності розробки ПС та адекватного оцінювання її якості без

залучення значних трудових ресурсів та підвищенням загального рівня якості кінцевого програмного продукту.

Таблиця 1.1 – Процедури, які необхідно автоматизувати при оцінюванні якості ПС

№ п/п	Складова процесу оцінювання якості ПС	Процедура, яку необхідно автоматизувати
1.	Проектування	Специфікація вимог
		Визначення пріоритетності атрибутів ПС
		Вибір метрик
		Визначення елементарних критеріїв оцінювання
		Процедура класифікації елементарних критеріїв оцінювання за глобальними критеріями
2.	Реалізація	Обчислення елементарних критеріїв якості ПС
		Обчислення частинних критеріїв якості ПС
		Обчислення глобальних критеріїв якості ПС
3.	Документування	Формування висновків за результатами оцінювання якості ПС та рекомендацій щодо її покращення

Обгрунтувавши необхідність автоматизації технологічних процесів на етапі розробки вимог до ПС та в процесі оцінювання їх якості потрібно дослідити наявні на ринку CASE-засоби, які орієнтовані на підтримку відповідних процедур.

Виконаємо аналіз технологій розроблення ПС та CASE-засобів, які їх підтримують на етапах ЖЦ. При цьому основну увагу зосередимо на етапі

розроблення вимог та оцінювання якості. На практиці використовують низку технологій розроблення ПС. Найширше використовуваними є такі:

- Microsoft Solutions Framework (MSF).
- Custom Development Method Oracle (CDMO).
- Rational Unified Process (RUP).

Ці технології базуються на двох основних підходах: структурному та об'єктно-орієнтованому. Технологія MSF є платформно-незалежною, яка орієнтована на розроблення ПС і розвиток інформаційної інфраструктури. Засоби цієї технології підтримують розподілені обчислення та застосування технології "клієнт-сервер".

Під час проектування вимог до ПС MSF використовує метод шаблонів та UML діаграм. Засобами графічного моделювання та документування вимог ПС є Microsoft Visio та пакет Microsoft Office. Microsoft Visio підтримує генерацію UML-діаграм, зокрема Use case діаграм. Для управління проектом та контролю загального процесу створення ПС використовують засіб Microsoft Project.

Технологія RUP базується на принципах об'єктно-орієнтованого підходу створення ПС та мови графічного моделювання UML. Підтримку цієї технології забезпечують засоби фірми IBM групи Rational. Для розроблення вимог за об'єктно-орієнтованим підходом використовують діаграми класів та Use case діаграми, що відображають функціональність майбутньої ПС. Засобом, що підтримує моделювання діаграм цього типу є Rational Rose. Інший засіб керування вимогами та їх документування – середовище Rational Requsite Pro. Основне призначення цього засобу полягає у наданні можливості відслідковування та зручного внесення змін у вимоги. При цьому вимоги представляють у текстовому вигляді з відображенням діаграм, змодельованих в Rational Rose.

Об'єктно-орієнтовний підхід розроблення вимог втілений також в автоматизованому засобі DOORS компанією Telelogic. Принцип

проектування вимог відображає структуру шаблону, що рекомендований у стандарті. В основі технології CDMO, лежить метод ORACLE CASE* METHOD, який базується на визначенні об'єктів (сутностей) та зв'язків між ними, що фактично є структурним підходом. Технологія CDM підтримується інструментальними засобами компанії Oracle і використовується під час створення автоматизованих інформаційних систем на основі реляційних баз даних. Для розроблення та керування вимогами використовується засіб автоматизації Oracle Designer.

Цей засіб дає змогу моделювати діаграми "сутність-зв'язок". З його допомогою можна лише визначати основні сутності всередині системи та зв'язки між ними, але неможливо описати систему загалом та процеси, які в ній відбуваються. В Oracle Designer відсутня можливість представлення вимог якості та обмежень, що істотно звужує область його застосування. Засобами автоматизації, які підтримують принципи структурного підходу щодо проектування ПС, є також програмні продукти ARIS Toolset, ERwin Datamodeler, Process Modeler (BPwin).

Виходячи з результатів проведеного аналізу можна стверджувати, що тільки технології MSF та DOORS мають засоби формалізованого представлення вимог якості, які використовують метод шаблонів. Однак структура і класифікація рубрик шаблонів не уніфіковані, тому у разі їх використання можуть виникати неоднозначності трактувань, що істотно звужує область застосування цих CASE-технологій.

Для перевірки відповідності готового програмного продукту заявленим у специфікації вимогам фірми-розробники використовують автоматизовані засоби тестування. Засоби, які б здійснювали автоматизацію технологічних операцій в процесі оцінювання якості ПС відсутні.

В зв’язку із зростаючою складністю програмних систем (ПС) стає все важче задовольняти вимоги якості при їх проектуванні. Для розв’язання цієї задачі з мінімальними втратами цей процес переносять на більш ранні стадії

проектування, а саме при проектуванні архітектури. Архітектура при цьому визначається як набір компонентів, які інкапсулюють логіку обчислень і зв'язки, які забезпечують взаємодію компонентів та створюють їх конфігурацію. Архітектура ПС забезпечує абстрактну модель високого рівня для представлення структури і ключових властивостей ПС і створює передумови забезпечення якості ПС.

Процес проектування архітектури включає декілька етапів [7]:

– визначення вимог до ПС, як функціональних, так і вимог якості, яке виконується на основі аналізу потреб всіх зацікавлених сторін. Також необхідно визначити відносну важливість атрибутів якості. Після цього необхідно провести комунікацію вимог якості до ПС на вимоги якості до архітектури;

– вибір альтернативних проектних рішень.

На основі аналізу вимог створюються альтернативні проектні рішення, які в подальшому будуть розглядатись для пошуку кращого з них. Для створення альтернативних архітектур повинна використовуватись технологія, базована на патернах.

– аналіз і оцінювання проектних рішень.

Кожен варіант проектного рішення повинен бути оцінений і порівняний з іншими. Архітектор повинен при цьому враховувати те, що альтернативи по різному впливають на реалізацію атрибутів якості, а атрибути, у свою чергу, мають різну відносну важливість. Оскільки вимоги до ПС можуть змінюватись як в процесі проектування, так і під час експлуатації, то будуть змінюватись і пріоритети атрибутів, що може вплинути на порядок ранжування альтернатив. Це також необхідно враховувати при виборі варіантів рішення;

– загальний архітектурний аналіз і прийняття рішення.

Використовуючи результати попереднього етапу, архітектор обирає найкращий варіант з точки зору задоволення всіх вимог якості. Якщо такого

варіанта архітектури немає, то досліджується конфлікти між критеріями якості і будуються області компромісів, на основі аналізу яких обирається рішення.

Приведемо короткий огляд існуючих методів оцінювання і вибору архітектури програмних систем з аналізом повноти реалізації в них наведених вище етапів.

1.2 Рекомендації по проектуванню багатошарових додатків

У даній частині роботи обговорюється загальна структура додатків з точки зору логічної угруповання компонентів в окремі шари, які взаємодіють один з одним і з іншими клієнтами і додатками. 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. У цьому розділі буде розглянуто, як розділяти додатки на логічні частини, як вибирати відповідну функціональну компоновку додатки і як забезпечити підтримку додатком безлічі типів клієнтів. Також ми розповімо про сервіси, які можуть використовуватися для надання логіки в шарах додатків.

Важливо розуміти різницю між шарами і рівнями. Шари (Layers) [7] описують логічну угруповання функцій і компонентів у додатку, тоді як рівні (tiers) описують фізичний розподіл функцій і компонентів по серверів, комп'ютерів, мережам або віддаленим местоположениям. Незважаючи на те, що і для шарів, і для рівнів застосовується одна і так само термінологія (подання, бізнес, сервіси та дані), варто пам'ятати, що тільки рівні подразумевают 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, трирівневе, n-рівневе.

1.2.1 Логічний поділ на шари

Незалежно від типу проектованого додатку і того, чи є в нього інтерфейс користувач або він є сервісним додатком, що просто надає послуги, його структуру можна розкласти на логічні групи програмних компонентів. Ці логічні групи називаються шарами. Шари допомагають розділити різні типи завдань, здійснювані цими компонентами, що спрощує створення дизайну, підтримуючого можливість повторного використання компонентів. Кожний логічний шар включає ряд окремих типів компонентів, згрупованих у 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 створюваного дизайну. Роздільна додатки на шари, що виконують різні ролі та 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.

1.2.2 Шар представлення, бізнес-шар і шар даних

На найвищому і найбільш абстрактному рівні логічне представлення архітектури системи може розглядатися як набір взаємодіючих компонентів, згрупованих в шари [7]. На рисунку 1.2 показано спрощене високорівневе представлення цих шарів та їх взаємовідносин з користувачами, іншими додатками, що викликають сервіси, реалізовані в бізнес-шарі додатки, джерелами даних, такими як реляційні бази даних або веб-сервіси, що забезпечують доступ до даних, і зовнішніми або віддаленими сервісами, використовуваними додатком.

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

Як показано з рисунку 1.2, додаток може складатися з ряду базових шарів. Типовий тришаровий дизайн, представлений на рисунку 1.2, включає наступні шари:

– 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, інкапсуліріванной в бізнес-шарі.

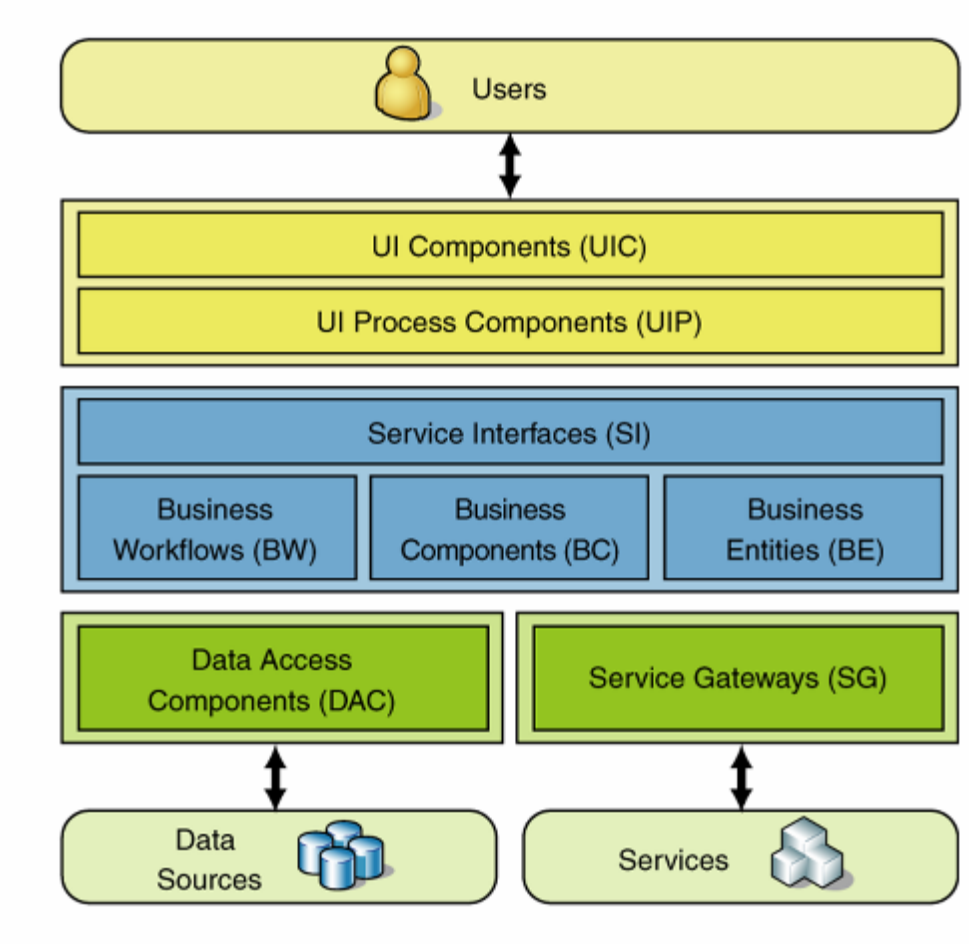


Рисунок 1.2 – Логічне подання багатошарової архітектури системи

– 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, наданим іншими мережевими системами. Доступ може здійснюватися через сервіси. Шар даних надає універсальні інтерфейси, які можуть використовуватися компонентами бізнес-шару.

1.2.3 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-яке інше 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.

1.2.4 Шар сервісів

Звичайним підходом при створенні програми, яке повинно забезпечувати сервіси для інших програм, а також реалізовувати безпосередню підтримку клієнтів, є використання шару сервісів, який надає доступ до бізнес-функціональності додатку (див. рис. 1.3.). Шар сервісів забезпечує альтернативне уявлення, що дозволяє клієнтам використовувати інший механізм для доступу до додатка.

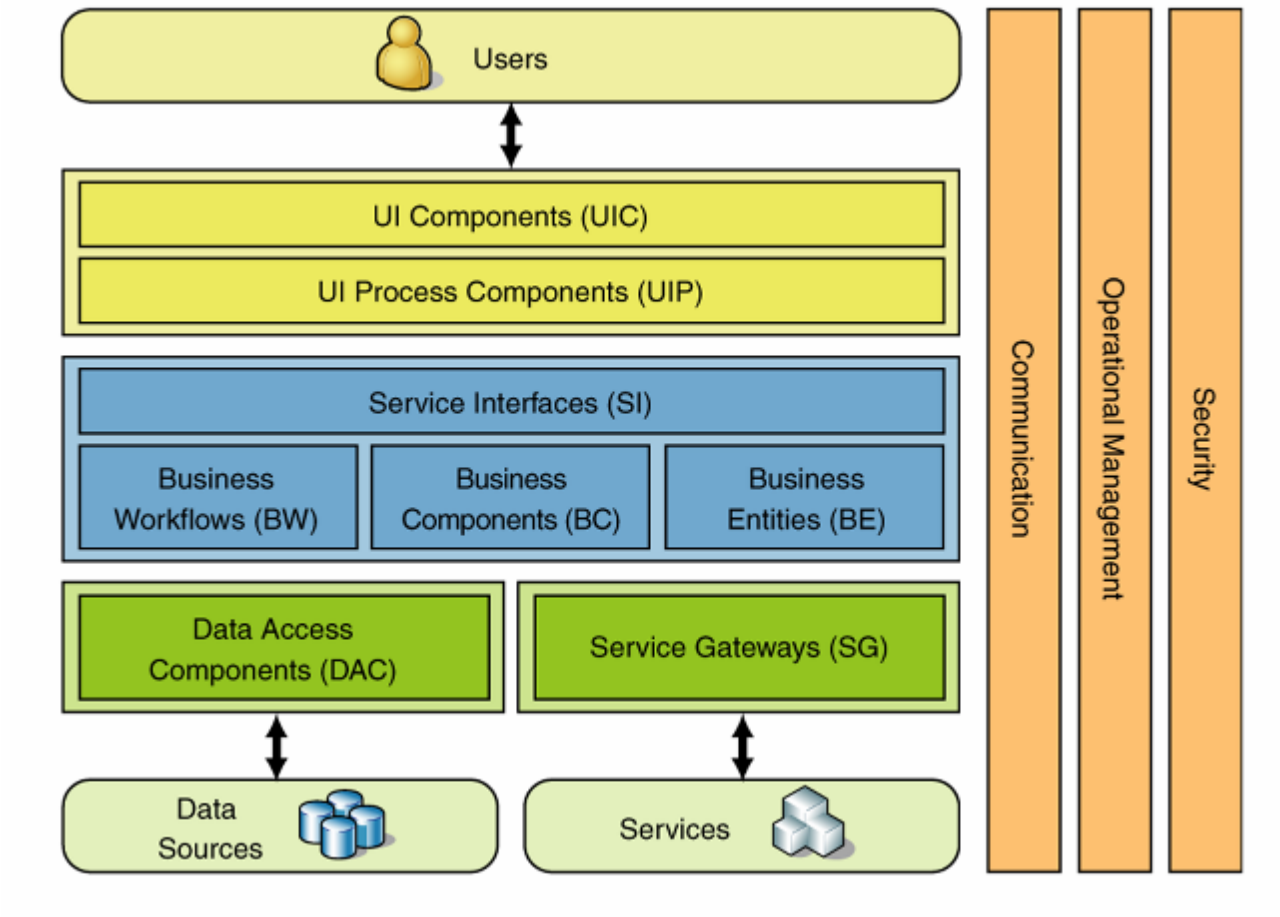


Рисунок 1.3 – Включення шару сервісів в додаток

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 програми для підтримки безлічі типів клієнтів, сприяє повторному використанню і більш високому рівню композиції функціональності в додатках.

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

1.2.5 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 та інтерфейси можуть змінюватися з часом, особливо у разі застосування гнучкої розробки, слідування цим етапам гарантовано забезпечить розгляд всіх важливих аспектів на початку процесу.

Звичайно при проектуванні використовується наступна послідовність кроків:

- Крок 1 – Вибір стратегії поділу на шари.
- Крок 2 – Вибір необхідних шарів.
- Крок 3 – Ухвалення рішення про розподіл шарів і компонентів.
- Крок 4 – З'ясування можливості згортання шарів.
- Крок 5 – Визначення правил взаємодії між шарами.
- Крок 6 – Визначення наскрізний функціональності.
- Крок 7 – Визначення інтерфейсів між шарами.
- Крок 8 – Вибір стратегії розгортання.
- Крок 9 – Вибір протоколів зв'язку.

1.3 Методика побудови архітектури та дизайну

Ітеративна техніка, яка може використовуватися при продумуванні і створення прототипу майбутньої архітектури. Вона допоможе звести воєдино ключові рішення, обговорювані в цьому посібнику, включаючи

рішення за параметрами якості, архітектурним стилям, типами додатків, технологіям і сценаріями розгортання.

Дана методика передбачає створення архітектури в ході процесу, що складається із серій по п'ять основних кроків кожна. У свою чергу, кожен крок розбитий на окремі аспекти, розглядом яких займається далі це керівництво. Ітеративний процес допомагає виробити можливі варіанти рішень, які в подальшому допрацьовуються в ході ітерацій і, в кінцевому рахунку, забезпечують створення дизайну архітектури, найбільш відповідної розроблюваного додатком. В кінці процесу можна створити огляд архітектури та представити його всім зацікавленим сторонам.

Залежно від підходу, використовуваного вашою організацією для розробки ПЗ, архітектура може багаторазово переглядатися в ході життєвого циклу проекту. Ця методика підходить для подальшої доробки архітектури, доповнення її новими аспектами, виявленими в наступний період збору відомостей, створення прототипів і фактичної розробки.

Тим не менш, важливо розуміти, що це всього лише один з можливих підходів. Існує безліч інших більш формальних методів визначення, аналізу та подання архітектури.

1.3.1 Вихідні дані, вихідні дані і етапи проектування

Вихідні дані проектування допомагають формалізувати вимоги та обмеження, які має реалізувати створювана архітектура. Зазвичай вихідними даними є варіанти використання і сценарії поведінки користувача, функціональні вимоги, нефункціональні вимоги (включаючи параметри якості, такі як продуктивність, безпека, надійність та інші), технологічні вимоги, цільова середу розгортання й інші обмеження.

У ході процесу розробки створюється список значущих з погляду архітектури варіантів використання, аспектів архітектури, які потребують спеціального уваги, і можливих архітектурних рішень, які задовольняють

вимогам і обмеженням, виявленим у процесі проектування. Загальною технікою поступової доопрацювання дизайну до тих пір, поки він не буде задовольняти всім вимогам і обмеженням, є ітеративна методика, що включає п'ять основних етапів, як показано на рисунку 1.4.

Цими етапами, які більш докладно розглядаються в наступних розділах, є:

1. Визначення цілей архітектури. Наявність чітких цілей допоможе зосередитися на архітектурі і правильному виборі проблем для вирішення. Точно позначені цілі допомагають визначити межі кожної фази: момент, коли завершена поточна фаза і все готово для переходу до наступної.

2. Основні сценарії. Використовуйте основні сценарії, щоб зосередитися на тому, що має першорядне значення, і перевіряйте можливі варіанти архітектур на відповідність цим сценаріями.

3. Загальне уявлення про додаток. Визначте тип програми, архітектуру розгортання, архітектурні стилі і технології, щоб забезпечити відповідність вашого дизайну реальним умовам, в яких буде функціонувати створюване додаток.

4. Потенційні проблеми. Виявити основні проблемні області на підставі параметрів якості і потреби в наскрізний функціональності. Це області, в яких найчастіше робляться помилки при проектуванні програми.

5. Варіанти рішень. У кожній ітерації повинен бути створений «пілот» або прототип архітектури, що є розвитком і доробкою рішення. Перш ніж переходити до наступної ітерації, необхідно переконатися у відповідності цього прототипу основними сценаріями, проблемам і обмеженням розгортання.

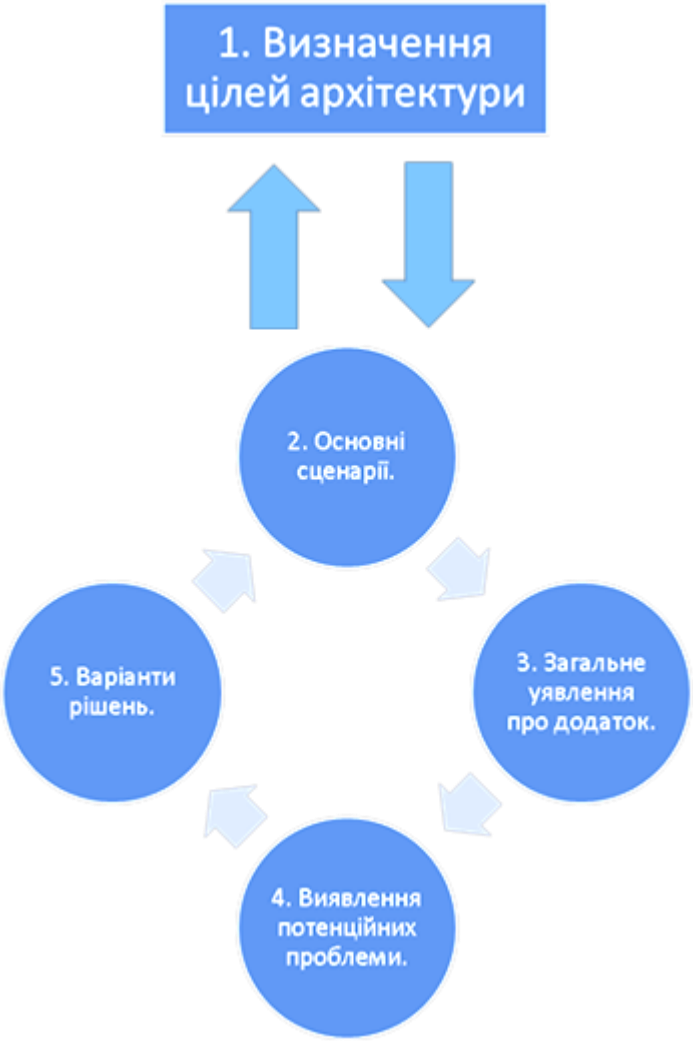


Рисунок 1.4 – Основні етапи ітеративного процесу проектування архітектури

Такий процес створення архітектури припускає ітеративний і інкрементний підхід. Спочатку створюється можливий варіант архітектури – узагальнений дизайн, який може тестуватися з основними сценаріями, вимогам, відомим обмеженням, параметрам якості і Архітектурної Базі. В ході доопрацювання варіанта архітектури, виявляються додаткові деталі і відомості про дизайн, результатом чого стає розширення основних сценаріїв, коректування загального подання додатка і підходу до вирішення проблем.

Не варто прагнути створити архітектуру за одну ітерацію. Кожна ітерація повинна розкривати додаткові деталі.

1.3.2 Визначення цілей архітектури

Мети архітектури [7] – це завдання і обмеження, що окреслюють архітектуру і процес проектування, що визначають обсяг робіт і які допомагають зрозуміти, коли пора зупинитися. Розглянемо ключові моменти у визначенні цілей архітектури:

– Початкова визначення завдань архітектури. Від цих завдань буде залежати час, що витрачається на кожну фазу проектування архітектури. Необхідно вирішити, що ви робите: створюєте прототип, проводите тестування можливих варіантів реалізації або виконуєте тривалий процес розробки архітектури для нового додатка.

– Визначення споживачів архітектури. Визначте, чи буде розробляється конструкція використовуватися іншими архітекторами, або вона призначається для розробників і тестувальників, IT-спеціалістів та керівників. Врахуйте потреби і підготовленість цільової аудиторії, щоб зробити розроблювану конструкцію максимально зручною для них.

– Визначення обмежень. Вивчіть всі опції і обмеження застосовуваної технології, обмеження використання та розгортання. Повністю розберіться з усіма обмеженнями на початку роботи, щоб не витрачати час або не стикатися з сюрпризами в процесі розробки програми.

1.3.3 Аналіз архітектури

Аналіз архітектури додатку [3,5,7] – критично важливе завдання, оскільки дозволяє скоротити витрати на виправлення помилок, якомога раніше виявити і виправити можливі проблеми. Аналіз архітектури слід виконувати часто: по завершенні основних етапів проекту і у відповідь на істотні зміни в архітектурі. Створюйте архітектуру, пам'ятаючи про

обгрунтування питаннях задаються при такому аналізі, це дозволить як поліпшити архітектуру, так і скоротити час, що витрачається на кожен аналіз.

Основна мета аналізу архітектури – підтвердження застосовності базової архітектури та її можливих варіантів, і також перевірка відповідності пропонованих технічних рішень функціональним вимогам і параметрам якості. Крім того, аналіз допомагає виявити проблеми і виявити області, потребують доопрацювання.

1.3.4 Ключові сценарії

Ключові сценарії [7] – це найбільш важливі сценарії для успіху створюваного додатка. Ключовий сценарій можна визначити як будь-який сценарій, який відповідає одному або більше з таких критеріїв:

- Він являє проблемну область – значну невідому область або область значного ризику.
- Він посилається на істотний для архітектури варіант використання (описується в наступному розділі).
- Він являє взаємодія параметрів якості з функціональністю.
- Він являє компроміс між параметрами якості.
- Наприклад, сценарії аутентифікації користувачів можуть бути ключовими сценаріями, тому що є перетинанням параметра якості (безпека) з важливою функціональністю (реєстрація користувача в системі). В якості іншого прикладу можна навести сценарій, заснований на незнайомій або нової технології.

1.3.5 Важливі з точки зору архітектури варіанти використання

Важливі з точки зору архітектури варіанти використання впливають на багато аспектів дизайну. Вони відіграють особливо важливу роль у забезпеченні майбутнього успіху створюваного додатка. Ці варіанти

використання важливі для приймання розгорнутого додатки і повинні охоплювати досить велику частину дизайну, щоб бути корисними при оцінці архітектури. До важливих з точки зору архітектури варіантам використання належать:

- Бізнес-критичний (Business Critical). Варіант використання, що має високий рівень використання або особливу важливість для користувачів або інших зацікавлених сторін, в порівнянні з іншими функціями, або передбачає високий ризик.
- Хто має великий вплив (High Impact). Варіант використання охоплює і функціональність, і параметри якості, або представляє наскрізну функцію, що має глобальний вплив на шари і рівні додатку. Прикладами можуть служити особливо вразливі з точки зору безпеки операції Create, Read, Update, Delete (CRUD).

Після виявлення важливих з точки зору архітектури варіантів використання вони можуть застосовуватися як засіб оцінки застосовності або незастосовність можливих варіантів архітектури додатку. Якщо варіант архітектури охоплює більше варіантів використання або описує існуючі варіанти використання більш ефективно, зазвичай це свідчить про те, що даний варіант архітектури є поліпшенням базової архітектури.

Хороший варіант використання буде збігатися з користувальницькою поданням, системним поданням і бізнес-виставою архітектури.

1.3.6 Загальне уявлення додатка

Необхідно створити загальне уявлення того, як буде виглядати готове додаток. Це загальне уявлення дозволить зробити архітектуру більш відчутній, зв'яже її з реальними обмеженнями та рішеннями. Створення загального уявлення додатка включає наступні дії:

1. Визначення типу програми. Перш за все, визначте, додаток якого типу створюється. Чи буде це мобільний додаток, насичений клієнт, насичене Інтернет-додаток, сервіс, Веб-додаток або деяке сполучення цих типів?

2. Визначення обмежень розгортання. При проектуванні архітектури додатку необхідно врахувати корпоративні політики та процедури, а також середовище, в якому планується розгортання програми. Якщо цільова середу фіксована або негнучка, конструкція додатки повинна відображати існуючі в цьому середовищі обмеження.

Також в конструкції додатки повинні бути враховані нефункціональні вимоги (Quality-of-Service, QoS), такі як безпека і надійність. Іноді необхідно поступитися чимось або в дизайні через обмеження в підтримуваних протоколах або топології мережі. Виявлення вимог та обмежень, присутніх між архітектурою додатки та архітектурою середовища на ранніх етапах проектування дозволяє вибрати відповідну топологію розгортання і вирішити конфлікти між додатком і цільової середовищем.

3. Визначення значущих архітектурних стилів проектування. Визначте, які архітектурні стилі будуть використовуватися при проектуванні. Архітектурний стиль – це набір принципів. Він може розглядатися як узагальнений шаблон, що забезпечує абстрактну базу для сімейства систем. Кожен стиль визначає набір правил, які задають типи компонентів, які можуть використовуватися для компоновки системи, типи відносин, застосовуваних у компонуванні, обмеження за способами компоновки і допущення про семантику компонування.

Архітектурний стиль покращує секціонування і сприяє можливості повторного використання дизайну завдяки наданню рішень часто зустрічаються проблем. Типовими архітектурними стилями є сервісно-орієнтована архітектура (Service Oriented Architecture, SOA), клієнт / сервер,

багатошарова, шина повідомлень і проектування на основі предметної області. Додатки часто використовують поєднання стилів.

4. Вибір відповідних технологій. Нарешті, на підставі типу програми та інших обмежень вибираємо відповідні технології і визначаємо, які технології будуть використовуватися в майбутній системі. Основними факторами є тип розроблюваного докладання, передбачувана топологія розгортання програми та бажані архітектурні стилі. Вибір технологій також залежить від політик організації, обмежень середовища, кваліфікації штату і т.д.

1.3.7 Відповідні технології проектування архітектури

При виборі технологій для використання при проектуванні необхідно звертати увагу на те, що забезпечить обраний архітектурний стиль, тип і основні параметри якості для програми. Розглянемо рекомендації, які допоможуть вибрати технології подання, реалізації та зв'язку, найбільш підходящі для кожного типу додатків на платформі Microsoft:

– Мобільні додатки. Для розробки програми для мобільних пристроїв можуть використовуватися технології шару уявлення, такі як .NET Compact Framework, ASP.NET для мобільних пристроїв і Silverlight для мобільних пристроїв.

– Насичені клієнтські програми. Для розробки додатків з насиченими UI, розгортаються та виконуваними на клієнті, можуть використовуватися поєднання технологій шару уявлення Windows Presentation Foundation (WPF), Windows Forms і XAML Browser Application (XBAP).

– Насичені клієнтські Інтернет-додатки (RIA). Для розгортання насичених UI в рамках Веб-браузера можуть використовуватися модуль Silverlight ™ або Silverlight в поєднанні з AJAX.

– Веб-додатки. Для створення Веб-додатків можуть застосовуватися ASP.NET WebForms, AJAX, Silverlight, ASP.NET MVC і ASP.NET Dynamic Data.

– Сервісні програми. Для створення сервісів, що надають функціональність зовнішнім споживачам систем і сервісів, можуть використовуватися Windows Communication Foundation (WCF) і ASP.NET Web services (ASMX).

1.3.8 Наскрізна функціональність

Наскрізна функціональність [4,7] – це аспекти дизайну, які можуть застосовуватися до всіх верств, компонентам і рівням. Також це ті області, в яких найчастіше робляться помилки, що мають великий вплив на дизайн. Наведемо приклади наскрізний функціональності:

– Аутентифікація і авторизація. Як правильно вибрати стратегію аутентифікації та авторизації, передачі ідентифікаційних даних між шарами і рівнями і зберігання посвідчень користувачів.

– Кешування. Як правильно вибрати техніку кешування, визначити дані, що підлягають кешуванню, де кешувати дані і як вибрати відповідну політику закінчення терміну дії.

– Зв'язок. Як правильно вибрати протоколи для зв'язку між шарами і рівнями, забезпечення слабкого зв'язування між шарами, здійснення асинхронного обміну даними та передачі конфіденційних даних.

– Управління конфігурацією. Як виявити дані, які повинні бути налаштованим, де і як зберігати дані конфігурації, як захищати конфіденційні дані конфігурації і як обробляти їх в серверній фермі або кластері.

– Управління винятками. Як обробляти і протоколювати виключення і забезпечувати повідомлення у випадку необхідності.

– Протоколювання і інструментірованіе. Як вибрати дані, що підлягають протоколюванню, як зробити протоколювання налаштованим, і як визначити необхідний рівень інструментірованія.

– Валідація. Як визначити, де і як проводити валідацію; як вибрати методики для перевірки довжини, діапазону, формату і типу; як запобігти і відхилити введення неприпустимих значень; як очистити потенційно зловмисний і небезпечний введення; як визначити і повторно використовувати логіку валідації на різних шарах і рівнях додатки.

1.3.9 Базова архітектура і можливі варіанти архітектури

Базова архітектура[5,7] описує існуючу систему, то як вона виглядає сьогодні. Для нового проекту вихідна базова архітектура – це перше високорівневе представлення архітектури, на підставі якого будуть створюватися можливі варіанти архітектури. Можливий варіант архітектури включає тип програми, архітектуру розгортання, архітектурний стиль, обрані технології, параметри якості і наскрізну функціональність.

На кожному етапі розробки дизайну будьте впевнені, що розумієте основні ризики та вживати заходів щодо їх скорочення, проводите оптимізацію для ефективної і раціональної передачі проектних відомостей і створюєте архітектуру, забезпечуючи гнучкість і можливість реструктуризації. Можливо, архітектуру доведеться змінювати кілька разів, використовувати декілька ітерацій, можливих варіантів і безліч пілотних архітектур.

Якщо можливий варіант архітектури є поліпшенням, він може стати базою для створення і тестування нових можливих варіантів.

Ітеративний і інкрементний підхід дозволяє позбутися великих ризиків спочатку, ітеративно формувати архітектуру і через тестування підтверджувати, що кожна нова базова архітектура є поліпшенням

попередньої. Наступні питання допоможуть протестувати новий варіант архітектури, отриманий на підставі «пілота» архітектури:

- Дана архітектура забезпечує рішення без додавання нових ризиків?
- Дана архітектура усуває більше відомих ризиків, ніж попередня ітерація?
- Дана архітектура реалізує додаткові вимоги?
- Дана архітектура реалізує важливі з точки зору архітектури варіанти використання?
- Дана архітектура реалізує аспекти, пов'язані з параметрами якості?
- Дана архітектура реалізує додаткові аспекти наскрізний функціональності?

1.3.10 Пілотні архітектури

Пілотна архітектура (architectural spike) – це тестова реалізація невеликої частини загального дизайну або архітектури додатку. Її призначення – аналіз технічних аспектів конкретної частини рішення для перевірки технічних припущень, вибору дизайну з ряду можливих варіантів і стратегій реалізації або іноді оцінка термінів реалізації.

Пілотні архітектури часто застосовуються в процесах гнучкого або екстремального проектування, але можуть бути дуже ефективним способом поліпшення і доробки дизайну рішення незалежно від підходу до розробки. Завдяки їх сфокусированности на основних частинах спільного проекту рішення, пілотні архітектури можуть використовуватися для вирішення важливих технічних проблем і для скорочення загальних ризиків і невизначеностей в дизайні.

Після завершення моделювання архітектури можна приступати до доопрацювання дизайну, плануванню тестів і поданням рішень іншим учасникам процесу. Керуйтеся наступними рекомендаціями:

При документуванні можливих варіантів архітектури та варіантів її тестування намагайтеся не захаращувати цей документ, що забезпечить простоту його оновлення. Такий документ може включати відомості про цілі, тип програми, топології розгортання, основних сценаріях і вимогах, технологіях, параметрах якості і тестах.

– Використовуйте параметри якості для визначення обрисів дизайну та реалізації. Наприклад, розробники повинні знати антішаблони для виявлених архітектурних ризиків і використовувати відповідні перевірені схеми для вирішення даних проблем.

– Діліться одержуваними відомостями з учасниками групи та іншими зацікавленими сторонами. До них можуть відноситися група розробки додатку, група тестування і адміністратори мережі або системні адміністратори.

1.4 Рекомендації по проектуванню компонентів

Компоненти є засобом ізоляції певних наборів функцій в елементах, які можуть поширюватися і встановлюватися окремо від іншої функціональності. Дана глава містить загальні рекомендації щодо створення компонентів і описує типи компонентів, зазвичай вживані в шарах додатків, проектованих з використанням багатошарового підходу, обговорюваного в цьому керівництві. Хоча, методики побудови компонентів зазвичай не залежать від структури програми.

Загальні рекомендації з проектування компонентів

Розглянемо загальні рекомендації проектування компонентів додатків:

– Застосовуйте принципи SOLID при проектуванні класів, що. Принципи SOLID – це:

• Принцип єдиності відповідальності (Single responsibility). Клас повинен відповідати тільки за один аспект.

• Принцип відкритості / закритості (Open / closed principle). Класи повинні бути розширюваними без необхідності доопрацювання.

• Принцип заміщення Лискова (Liskov substitution principle). Підтипи і базові типи повинні бути взаємозамінні.

• Принцип відділення інтерфейсу (Interface segregation principle). Інтерфейси класів повинні бути клієнт-специфічними і вузьконаправленими. Класи повинні надавати різні інтерфейси для клієнтів, що мають різні вимоги до інтерфейсів.

• Принцип інверсії залежностей (Dependency inversion principle).

Залежності між класами повинні замінюватися абстракціями, що забезпечить можливість проектування зверху вниз без необхідності проектування спочатку модулів нижнього рівня. Абстракції не повинні залежати від деталей – деталі повинні залежати від абстракцій.

– Проектуйте сильно зв'язні компоненти. Не перевантажуйте компоненти введенням у них непов'язані або змішаної функціональності.

Наприклад, завжди уникайте змішування в компонентах бізнес-шару логіки доступу до даних і бізнес-логіки. Забезпечивши зв'язність функціональності, можна створювати збірки, що включають більше одного компонента, і встановлювати компоненти у відповідних шарах додатки, навіть якщо ці шари розділені фізично.

– Компонент не повинен залежати від внутрішніх деталей інших компонентів.

Кожен компонент або об'єкт повинен викликати метод іншого об'єкта або компонента, і цей метод повинен знати, як обробляти запит і, якщо необхідно, як направити його до відповідних подкомпоненте або інших

компонентів. Такий підхід дозволяє створювати більш адаптуються і зручні в обслуговуванні додатка.

– Продумайте, як компоненти будуть взаємодіяти один з одним.

Для цього потрібно розуміти, які сценарії розгортання повинно підтримувати створюване додаток, чи воно підтримувати взаємодію через фізичні кордону або межі процесу, або всі компоненти будуть виконуватися в одному процесі.

– Не змішуйте код наскрізний функціональності і прикладну логіку додатку.

Код, який реалізує наскрізну функціональність – це код, пов'язаний з безпекою, зв'язком або управлінням, таким як протоколюванням і інструментуванням. Змішання коду, що реалізує ці функції, з логікою компонентів може призвести до створення погано розширюваного і складного в обслуговуванні дизайну.

– Застосовуйте основні принципи компонентного архітектурного стилю. Ці принципи полягають у тому, що компоненти повинні бути придатними для повторного використання, замінними, розширюваними, інкапсульованими, незалежними і не залежати від контексту.

1.5 Розподіл компонентів по шарах

Кожен шар додатки містить набори компонентів, що реалізують функціональність даного шару. Ці компоненти повинні бути зв'язковими і слабо пов'язаними, щоб забезпечити можливість повторного використання і спростити обслуговування. На рисунках 1.2 та 1.3можна побачити, які типи компонентів зазвичай використовуються в кожному з шарів.

1.5.1 Компоненти шару представлення

Компоненти шару представлення реалізують функціональність, необхідну для забезпечення взаємодії користувачів з додатком. Зазвичай в шарі представлення розташовуються наступні типи компонентів:

– Компоненти для інтерфейсу користувача.

Конкретна реалізація користувальницького інтерфейсу додатку Інкапсульована в компоненти користувальницького інтерфейсу (UI). Це візуальні елементи програми, які використовуються для відображення даних користувачеві і прийому користувальницької введення. Компоненти UI, спроектовані для реалізації шаблону Separated Presentation, іноді називають Уявленнями (Views).

У більшості випадків їх роль полягає в наданні користувачеві інтерфейсу, який забезпечує найбільш відповідне подання даних і логіки додатка, а також в інтерпретації користувальницької введення і передачі його в компоненти логіки уявлення, які визначають вплив введення на дані і стан програми.

У деяких випадках в компонентах користувальницького інтерфейсу може міститися спеціальна логіка реалізації інтерфейсу користувача, однак, як правило, вони включають мінімальний обсяг логіки додатка, оскільки це може негативно позначитися на зручності обслуговування і можливості повторного використання, а також ускладнити модульне тестування.

– Компоненти логіки представлення.

Логіка представлення – це код додатку, що визначає поведінку і структуру програми таким чином, що вони не залежать від будь-якої конкретної реалізації інтерфейсу користувача. Компоненти логіки уявлення, головним чином, забезпечують реалізацію варіантів використання додатка (або користувальницьких історій) і координують взаємодії користувача з базовою логікою і станом додатки незалежно від UI.

Також вони відповідають за організацію надходять з бізнес-шару даних у формат, придатний для споживання компонентами UI. Наприклад, вони можуть агрегувати дані з багатьох джерел і перетворювати їх для більшої зручності відображення. Компоненти логіки подання можна поділити на дві категорії:

– Компоненти Presenter, Controller, Presentation Model і ViewModel.

Дані типи компонентів використовуються при реалізації шаблону Separated Presentation і часто інкапсулюють логіку уявлення шару уявлення. Щоб забезпечити максимальні можливості повторного використання і зручність тестування, ці компоненти не прив'язані до жодного конкретного класу, елементу або елементу управління UI.

– Компоненти сутностей уявлення.

Ці компоненти інкапсулюють бізнес-логіку і дані і спрощують їх споживання для користувача інтерфейсом і компонентами логіки уявлення, наприклад, шляхом перетворення типів даних або агрегації даних з декількох джерел. У деяких випадках, це бізнес-сутності бізнес-шару, використовувані безпосередньо шаром уявлення.

В інших випадках, вони можуть представляти підмножина компонентів бізнес-сутностей і створюватися спеціально для підтримки шару подання додатка. Сутності уявлення допомагають забезпечити несуперечність і дійсність даних в шарі уявлення. У деяких шаблонах роздільного уявлення ці компоненти називають моделями.

1.5.2 Компоненти шару сервісів

Додаток може надавати шар сервісів для взаємодії з клієнтами або використання іншими системами. Компоненти шару сервісів забезпечують іншим клієнтам і додаткам спосіб доступу до бізнес-логікою додатки і

використовують функціональність програми шляхом обміну повідомленнями по каналу зв'язку. Зазвичай в шарі сервісів розташовуються наступні типи компонентів:

– Інтерфейси сервісів.

Сервіси надають інтерфейс сервісів, в який передаються всі вхідні повідомлення. Опис набору повідомлень, якими необхідно обмінюватися з сервісом для здійснення ним певної бізнес-завдання, називається контрактом. Інтерфейс сервісу можна розглядати як фасад, що надає потенційним споживачам бізнес-логіку, реалізовану в додатку (як правило, це логіка бізнес-шару).

– Типи повідомлень.

При обміні даними в шарі сервісів структури даних укладені в структури повідомлень, що підтримують різні типи операцій. Наприклад, існують такі типи повідомлень, як Command (Команда), Document (Документ) та інші. Типи повідомлень – це контракти повідомлень, які використовуються для взаємодії споживачів і провайдерів сервісу. Також шар сервісів зазвичай надає типи даних і контракти, які визначають типи даних, використовувані в повідомленнях, і ізолюють внутрішні типи даних від даних, що містяться в типі повідомлення. Це запобігає розкриття внутрішніх типів даних зовнішнім споживачам, що могло б призвести до складнощів з контролем версій інтерфейсу.

1.5.3 Компоненти бізнес-шару

Компоненти бізнес-шару реалізують основну функціональність системи і інкапсулюють відповідну бізнес-логіку.

Бізнес-шар зазвичай включає наступні типи компонентів:

–Фасад додатку.

Цей необов'язковий компонент зазвичай забезпечує спрощений інтерфейс для компонентів бізнес-логіки часто шляхом об'єднання безлічі

бізнес-операцій в одну, що спрощує використання бізнес-логіки й скорочує кількість залежностей, оскільки зовнішнім зухвалим сторонам немає необхідності знати деталі бізнес-компонентів і відносини між ними.

–Компоненти бізнес-логіки.

Бізнес-логіка – це логіка додатки, пов'язана з витяганням, обробкою, перетворенням і управлінням даними додатка; застосуванням бізнес-правил і політик та забезпеченням несуперечності і дійсності даних. Щоб забезпечити найкращі умови для повторного використання, компоненти бізнес-логіки не повинні включати поведінку або логіку програми, пов'язані до конкретного варіанту використання або користувальницької історії. Компоненти бізнес-логіки можна розділити на наступні дві категорії:

–Компоненти робочого процесу.

Після того як дані введені в компоненти и1 і передані в бізнес-шар, додаток може використовувати їх для виконання бізнес-процесу. Багато бізнес-процеси складаються з безлічі етапів, які повинні здійснюватися у відповідному порядку і можуть взаємодіяти один з одним за допомогою механізмів координування. Компоненти робочого-процесу визначають і управляють тривалими багатоетапними бізнес-процесами і можуть бути реалізовані з використанням інструментів управління бізнес-процесами. Компоненти робочого процесу працюють з компонентами бізнес-процесу, які створюють екземпляри компонентів робочого процесу та здійснюють операції з ними.

–Компоненти бізнес-сутностей.

Бізнес-сутності, або, більш узагальнено, бізнес-об'єкти, інкапсулюють бізнес-логіку і дані, необхідні для подання в додатку елементів реального світу, таких як замовники (Customers) або замовлення (Orders). Вони зберігають значення даних і надають їх через властивості; містять і керують бізнес-даними, які використовуються додатком; і забезпечують програмний доступ із збереженням стану до бізнес-даних і відповідної функціональності.

Також бізнес-суті проводять перевірку містяться в них даних і інкапсулюють бізнес-логіку для забезпечення несуперечності даних і реалізації бізнес-правил і поведінки.

Дуже часто бізнес-сутності повинні бути доступними компонентам і сервісів як бізнес-слоя, так і шару даних. Наприклад, бізнес-сутності можуть зіставлятися з джерелом даних, і до них можуть виконувати доступ бізнес-компоненти. Якщо шари розташовуються на одному рівні, бізнес-сутності можуть використовуватися спільно безпосередньо через покажчики.

Однак при цьому все одно має бути забезпечений поділ бізнес-логіки і логіки доступу до даних. Цього можна досягти шляхом переміщення бізнес-сутностей в окрему збірку, доступну для використання збірками та бізнес-сервісів, і сервісів даних. Цей підхід аналогічний використанню шаблону інверсії залежностей, коли бізнес-суті відокремлюються від бізнес-шару і шару даних, і їх залежність від бізнесутностей реалізується, як спільно використовуваний контракт.

1.5.4 Компоненти шару доступу до даних

Компоненти шару доступу до даних забезпечують доступ до даних, розміщеним в рамках системи, і до даних, що надаються іншими мережевими системами. Звичайно шар доступу до даних включає наступні типи компонентів:

–Компоненти доступу до даних.

Ці компоненти абстрагують логіку, необхідну для доступу до базових сховищ даних. Для більшості завдань доступу до даних необхідна загальна логіка, яка може бути виділена і реалізована в окремих допоміжних компонентах, доступних для повторного використання, або підходящої допоміжної інфраструктурі. Це може спростити компоненти доступу до даних і централізувати логіку, що полегшує обслуговування. Решта завдань,

загальні для компонентів шару даних і не відносяться ні до одного набору компонентів, можуть бути реалізовані як окремі службові компоненти. Допоміжні та службові компоненти часто об'єднуються в бібліотеку або інфраструктуру, що полегшує їх повторне використання в інших додатках.

– Агенти сервісів.

Якщо бізнес-компонент повинен використовувати функціональність, що надається зовнішнім сервісом, ймовірно, буде потрібно реалізувати код для управління семантикою взаємодії з конкретним сервісом. Агенти сервісів ізолюють спеціальні аспекти виклику різних сервісів в додатку і можуть забезпечувати додаткові сервіси, такі як кешування, підтримка роботи в автономному режимі і базове зіставлення форматів даних, що надаються сервісом, і форматів, необхідним додатком.

1.5.5 Компоненти наскрізний функціональності

Деякі завдання необхідно виконувати в багатьох шарах. Компоненти наскрізний функціональності реалізують спеціальні типи функціональності, доступ до яких можуть здійснювати компоненти будь-якого шару. Розглянемо основні типи компонентів наскрізний функціональності:

– Компоненти для реалізації безпеки. Сюди відносяться компоненти, що здійснюють аутентифікацію, авторизацію і валідацію.

– Компоненти для реалізації завдань операційного управління. Сюди відносяться компоненти, що реалізують політики обробки виключень, протоколювання, лічильники продуктивності, конфігурацію і трасування.

– Компоненти для реалізації взаємодії. Сюди відносяться компоненти, які взаємодіють з іншими сервісами та додатками.

1.6 Графічне представлення архітектури

Важливо графічно представити розроблювану архітектуру. Спочатку це може бути наближене зображення, зроблене від руки (див. рис. 1.5).

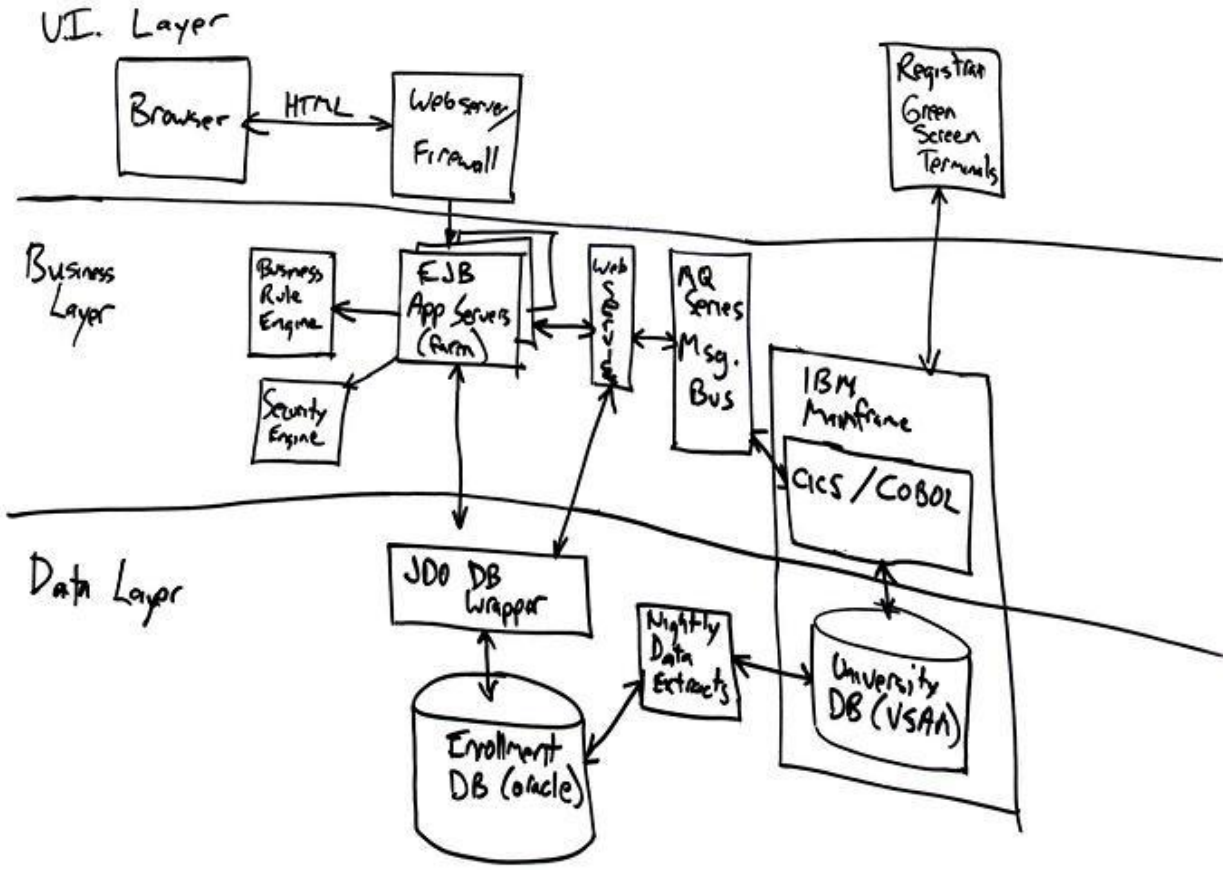


Рисунок 1.5 – Приклад графічного представлення дизайну веб-додатку в першому наближенні із зазначенням протоколів і методів аутентифікації, які передбачається використовувати

Незалежно від того, чи робиться це на папері, у вигляді слайдів або в іншому форматі, головне – показати основні обмеження і прийняті рішення для того, щоб позначити межі і почати обговорення. Насправді, це має подвійну цінність. Якщо неможливо наочно уявити архітектуру, значить, немає повного її розуміння. Якщо можливо зобразити чітку і коротку

діаграму, вона буде зрозумілою іншим, і буде набагато простіше пояснювати деталі.

1.6.1 Основні проблеми при проектуванні архітектури

Визначити основні потенційні проблеми архітектури додатку, щоб зрозуміти області, в яких найбільш ймовірне виникнення помилок. До потенційних проблем відносяться поява нових технологій і критично важливі бізнес-вимоги. Наприклад, «Чи можу я переходити з одного сервісу стороннього виробника до іншого?», «Чи можу я додати підтримку нового типу клієнта?», «Чи можу я швидко змінювати бізнес-правила оплати послуг?» І «Чи можу я перейти до нової технології для компонента X?». Незважаючи на те, що це вкрай узагальнені аспекти, як правило, при реалізації вони (та інші зони ризику) проектуються в параметри якості і наскрізну функціональність.

1.6.2 Параметри якості

Параметри якості [2,11-14] – це загальні властивості архітектури, які впливають на поведінку під час виконання, дизайн системи і взаємодія з користувачем. Та ступінь, з якою додаток забезпечує необхідне сполучення параметрів якості, таких як зручність і простота використання, продуктивність, надійність і безпеку, визначає успішність дизайну і загальну якість програмного продукту. При проектуванні програми, відповідального будь-якого з цих параметрів, необхідно врахувати вплив та інших вимог, повинні бути проаналізовані плюси і мінуси по відношенню до інших параметрах якості. Важливість або пріоритетність кожного з параметрів якості для різних систем різна. Наприклад, для бізнес-додатки (line-of-business, LOB) продуктивність, масштабованість, безпеку і зручність використання будуть більш важливі, ніж можливість взаємодії з іншими

системами. А ось для коробкового додатки така можливість буде мати більше значення, ніж для ШБ-додатки.

Параметри якості представляють функціональні області, які потенційно можуть впливати на всі додаток, на всі його верстви та рівні. Деякі параметри ставляться до всього дизайну системи, тоді як інші стосуються тільки часу виконання, часу проектування або взаємодії з користувачем. Наступний список систематизує відомості про параметри якості і допомагає зрозуміти, на які сценарії їх вплив найбільш ймовірно:

– Загальносистемні якості. Загальні якості системи в цілому, такі як можливість технічної підтримки та тестової.

– Якості часу виконання. Якості системи, притаманні безпосередньо під час виконання, такі як доступність, можливість взаємодії з іншими системами, керованість, продуктивність, надійність, масштабованість і безпеку.

– Конструктивні якості. Якості, що відображають дизайн системи, такі як концептуальна цілісність, гнучкість, зручність і простота обслуговування і можливість повторного використання.

– Користувальницькі якості. Зручність і простота використання системи.

1.7 Огляд методів оцінювання і вибору архітектури пс на основі вимог якості

Існує раннє і пізнє оцінювання архітектур. Раннє оцінювання використовується тоді, коли ще не створено програмних компонентів або їх моделей. Таке оцінювання базується на досвіді розробників та логічному обґрунтуванні, оскільки відсутні артефакти, які дають змогу імітувати роботу ПС. Методи, які реалізують раннє оцінювання, базуються на сценаріях. До цих методів належать наступні: SAAMі ATAM. В методі SAAMдля

коректного порівняння архітектур, існуючих та тих, що розглядаються, запропоновано аналізувати їх у трьох аспектах, а саме – функціональність, структура та розміщення. На основі пріоритетів зацікавлених сторін визначаються критерії якості. Для перевірки задоволення кожного атрибута якості розробляється сценарій і проводиться оцінка рівня задоволення даного атрибуту варіантом архітектури.

ATAM (ArchitectureTrade-OffAnalysisMethod) – метод в якому оцінюються ризики того, що архітектура не задовольняє концептуальним вимогам, які описуються сценарієм. Метод ATAM подібний до SAAM, але в ньому на основі аналізу сценаріїв для відібраних архітектур проводиться оцінка ризиків задоволення атрибутів якості. Оцінку ризиків проводить група експертів, яка також ранжує альтернативні варіанти за рівнем ризику і визначає так звані точки чутливості у компонентах чи зв'язках архітектури, також аналізуються компроміси між критеріями якості.

Методи ATAMі SAAM поєднані єдиною концепцією і часто використовуються в сукупності.

Вимоги якості до архітектури в даних методах визначаються експертами, не використовуються формальні методи. Тому має місце суттєвий вплив суб'єктивних факторів і відсутні методи автоматизації цих процесів.

Аналіз проектних рішень відбувається послідовно по одному атрибуту якості, при виборі варіанта архітектури не використовуються методи оптимізації. Рівень автоматизації процесів низький через недостатнє використання формальних методів.

Для обґрунтованого вибору рішення в методі SAAM/ATAMвибрані альтернативні архітектури аналізуються на ефективність витрат методом CBAM. Цей метод забезпечує економічний аналіз ПС, яка базується на вибраних в попередніх методах варіантах архітектури та сценаріях моделювання. Експерти призначають оцінки критеріям якості в балах від 1

до 100 і ранжують архітектури за значенням, яке ці архітектурні рішення забезпечують для атрибуту якості. Оцінка кожного варіанта архітектури обчислюється за формулою:

$$B(A_i) = \sum_{j=1,K} \left(Cont_{i,j} \cdot Q_j \right) \; i = \overline{1,n}. \tag{1.1}$$

Тут $Cont_{ij}$ – вага і-ї архітектури відносно j-го атрибута;

Q_j – пріоритет j-го атрибута.

Метод забезпечує оцінку затрат на реалізацію кожної альтернативи і дає можливість обчислити показник бажаності як відношення прибутку до затрат. На основі отриманих даних проводиться вибір кращого рішення.

Метод CBAM використовує архітектурні рішення і атрибути якості, отримані із SAAM/ATAM, а забезпечує лише оцінку рішень, тобто фактично реалізує третій і частково четвертий етапи проектування архітектури.

Часто виникають задачі створення ПС на базі існуючої шляхом перепроектування для задоволення нових вимог якості. Для вирішення таких задач було створено метод реінжинірингу архітектури ПС на основі сценаріїв SSAR [3], який є сукупністю чотирьох методів оцінки архітектур відносно атрибутів якості:

- оцінка на основі сценаріїв;
- моделювання;
- математичне моделювання;
- оцінка на базі практичного досвіду.

При використанні SSARобирається один із методів, але основним є метод оцінювання на основі сценаріїв. Цей метод подібний до того, що реалізується в SAAM.

При використанні моделювання основні компоненти ПС реалізуються в коді, а інші моделюються комп’ютером, утворюючи виконувану систему.

При використанні математичного моделювання характеристики якості ПС оцінюються за допомогою математичних моделей операцій, на яких ці характеристики реалізуються.

Оцінювання на базі практичного досвіду дає можливість виявити дефекти проектних рішень та проблеми, які необхідно усунути.

Метод SSAR не містить процедур вибору альтернативних архітектур, а також виявлення конфліктів і пошук компромісів між атрибутами якості. Оцінювання проводиться послідовно по кожному атрибуту якості без використання процедури оптимізації. Спільним недоліком розглянутих методів є послідовне оцінювання архітектури по одному параметру, що робить процес вибору трудомістким і неформалізованим. Тому поява робіт, в яких було використано процедуру аналізу ієрархій, дозволив значно покращити процес вибору архітектури і формалізувати його.

В методі SAHR[9] використовується порівняльне оцінювання альтернатив стосовно реалізації атрибутів якості. Він дає змогу визначити відносні ваги альтернатив по кожному атрибуту якості і проранжувати їх. За призначеними зацікавленими сторонами пріоритетами атрибутів якості обчислюється їх усереднене значення і визначаються ваги альтернатив відносно сукупності атрибутів якості.

Отримані відносні оцінки альтернатив можуть використовуватись для аналізу конфліктів між атрибутами якості і пошуку компромісного рішення.

Перевагами методу SAHRє оцінювання альтернатив по всіх атрибутах якості, оптимізація рішень та досить високий рівень формалізації, що дає змогу автоматизувати процес.

З проведеного аналізу слідує, що методи оцінювання архітектур базуютьсяв основному на експертній інформації. При цьому широко використовуються знання та досвід проектувальників. Тому для підвищення ефективності цих методів необхідно використовувати їх у складі експертної системи, в якій знання формалізовані в базі знань, а процеси введення та

обробки експертної інформації автоматизовані з допомогою апаратно-програмної платформи.

Метод аналізу ієрархій Сааті [2,9], дозволяє отримати порівняльні оцінки множини альтернатив по задоволенню критеріїв якості. Суттєвим недоліком застосування MAI є обмежена кількість альтернатив, які можна оцінювати одночасно (n<=7±2), що викликано неузгодженістю елементів матриць парних порівнянь. Для вирішення цієї проблеми, в запропонована модифікація MAI, в якій вагові множники альтернатив визначаються з умови мінімізації неузгодженості матриці парних порівнянь, що приводить вихідну задачу до задачі математичного програмування.

В розділі розглянуті питання застосування модифікованого MAI (MMAI) до задачі вибору оптимальної архітектури програмних систем. Отримані відносні оцінки альтернатив в MMAI можуть використовуватись для аналізу конфліктів між атрибутами якості і пошуку компромісного рішення при виборі архітектури по множині критеріїв якості. Для остаточного вибору варіанта архітектури з врахуванням сукупності критеріїв обчислюють значення інтегрального критерію у вигляді скалярної згортки, для чого необхідно задати коефіцієнти пріоритетів критеріїв якості.

Але пріоритети критеріїв різних груп фахівців суттєво різняться, тому для отримання компромісного результату необхідно проводити додаткові дослідження. Також важливо при виборі архітектури враховувати чутливість отриманого ранжування альтернатив до зміни пріоритетів критеріїв якості, викликаною зміною вимог предметної області.

2 МЕТОДИ ТА ЗАСОБИ, НЕОБХІДНІ ДЛЯ РОЗВ'ЯЗКУ ЗАДАЧІ

2.1 Загальна ідея функціонування програмної системи

Для того, щоби представити архітектуру будь-якої програмної системи, скористаємось концепцією, запропонованою корпораціє\ю Microsoft [7], де програмна архітектура повинна подаватись у багатошаровому поданні. Згідно цієї пропозиції на кожному рівні архітектури застосовуються власні компоненти (патерни), котрі забезпечують функціональність саме свого рівня і взаємодіють з компонентами сусідніх рівнів. Для зручності класифікації та вибору патерни згруповані по категоріях чи модулях. Кожен модуль функціонально вирішує задачі певного роду на загальному рівні абстракції. В конкретних умовах такі компоненти можуть бути адаптовані до своєї предметної області.

Таким чином, будь-який програмний проектований продукт може бути представлений у вигляді декількох частин, кожна з яких відповідає своєму шарові в архітектурі. Згрупувавши задачі, котрі має вирішувати проектована система, для кожного шару вибираються свої компоненти з наявного набору. При чому для кожної типової задачі існує декілька компонентів. Через це можна зкомпонувати декілька рішень для одного і того ж функціоналу, але у вигляді різних архітектурних проектів. Тобто матимемо декілька програмних архітектур (ПА).

В дипломній роботі пропонується автоматизувати процес проектування архітектури програмного забезпечення шляхом створення експертної системи. При чому компоновка архітектури буде виконуватись за принципом заповнення фрейму. Яким чином на основі стандартних шаблонів (патернів) проектування можна буде сформувати декілька альтернативних архітектур, в кожній з яких складовими частинами будуть стандартні патерни.

Особа, що займається розробкою програмної архітектури (ПА), виконує поділ функціоналу програми по шарах. Після цього для кожного шару здійснюється підбір патернів і таким чином заповнюються згадані вище фрейми, кожен з яких представляє своє архітектурне рішення. В результаті буде декілька таким чином заповнених фреймів, кожен з яких представляє архітектурне рішення. В загальному випадку таких рішень може бути довільна кількість, яка обмежується максимальною кількістю комбінацій стандартних патернів проектування.

Отже, перейдемо до опису процесу проектування експертної системи для автоматизованого проектування та оцінювання програмних архітектур. Головним фактором, котрий мав визначальний вплив на підхід до проектування такої системи, є те, що вона повинна представляти собою експертну систему (систему підтримки прийняття рішень) в області оцінювання програмних архітектур, а саме альтернативних рішень для однієї системи з наступним фінальним вибором найоптимальнішого проекту ПА [1]. Таким чином, перерахуємо основні функціональні вимоги до такої системи:

– експерти з проектування ПА повинні мати можливість виконати порівняльне оцінювання альтернативних архітектурних рішень;

– система повинна розмежовувати функціональні можливості користувачів на ролі адміністратора системи та користувачів з роллю архітектора ПА;

– проектована система повинна бути масштабованою по відношенню до випадків, коли потрібно змінювати кількість шарів архітектури та/або набір патернів проектування;

– доступ до елементів репозиторію програмних архітектур повинен надаватись на основі привілеїв, розподілених кожній ролі.

2.2 Опис методу створвєння монжини альтрєантивних рішєнь ПА

Коротко подамо процес утворення для випадку генерації декількох альтернатив однієї ПА. Основним і першим кроком в проектуванні архітектури є формування каркасу програмного продукту, котрий реалізуватиме основний функціонал. Як зазначалось вище, цей каркас має бути розподілений по умовних шарах, а вже наступним кроком буде заповнення цих шарів конкретними патернами проектування. Знову ж таки, для кожного шару знайдеться декілька патернів, через що і ортримаємо множину альтернативних архітектурних рішень. Для прикладу можна згадати загальноприйняті підходи, тобто патерни на найвищому рівні абстракції, реалізовані у багатьох фреймворках:

- MVC (Model-view-controller).
- MVP (Model-View-Presenter) [6].

Ці два патерни, наведені в якості прикладу, реалізують однаковий набір функціональних вимог, але спроектовані по різному – мають різну структуру як з функціональної точки зору, так і з точки зору внутрішньої структури і взаємодії компонентів. Якщо далі заглибитись в деталі, то шаблон проектування MVP є дочірнім по відношенню до MVC.

В якості аналогічного прикладу можна також згадати шаблони проектування, котрі використовуються для взаємодії з базами даних. Наприклад в рамках фреймворку .NET можна серед інших використовувати для підключення до баз даних бібліотеку (та фреймворк) DAO (data access object) або ж скористатись нативним API для конкретної СКБД і використовувати її "рідні" функції, жертвуючи при цьому властивостями гнучкості, переносимості та адаптованості системи.

Тобто за рахунок використання абстрактних класів бібліотеки DAO досягається краща гнучкість, захищеність та незалежність від конкретної СКБД. В той самий час пряма адресація на основі нативного API дозволяє

швидко виконати прямі запити до вибраної бази даних (БД), але не найкращим рішенням для випадків, коли система має бути орієнтовнаа на використання різних СКБД.

Для прикладу розглянемо випадок, коли на кожен шар обрано однакові компоненти для двох архітектурних альтернатив, за винятком патернів підключення до БД та патернів інтерфейсу користувача. Такий приклад показаний на рис. 2.2. Таким чином через просту комбінацію цих патернів ми отримаємо чотири альтернативних рішення, що відображено на рис. 2.3.

2.3 Метод для здійсненя порівняння архітектурних альтернатив

Після того, ак альтернативні архітектурні рішення створені, вони готові до порівняння. Таке порівняння здійснюється на основі експертних технологій шляхом попарного співтсавлення альтенативних архітектур стосвоно того, яка з них краще реалізує той чи інший атрибут якості. В результаті такого співставлення отримуємо матрицю парних порівняь (МПП) для кожного критерію, як показано на рисунку 2.2.

Альтернативи для порівняння з рис. 2.3 виглядають наступним чином:

- MVC – Пряма адресація.
- MVC – DAO.
- MVP – Пряма адресація.
- MVP – DAO.

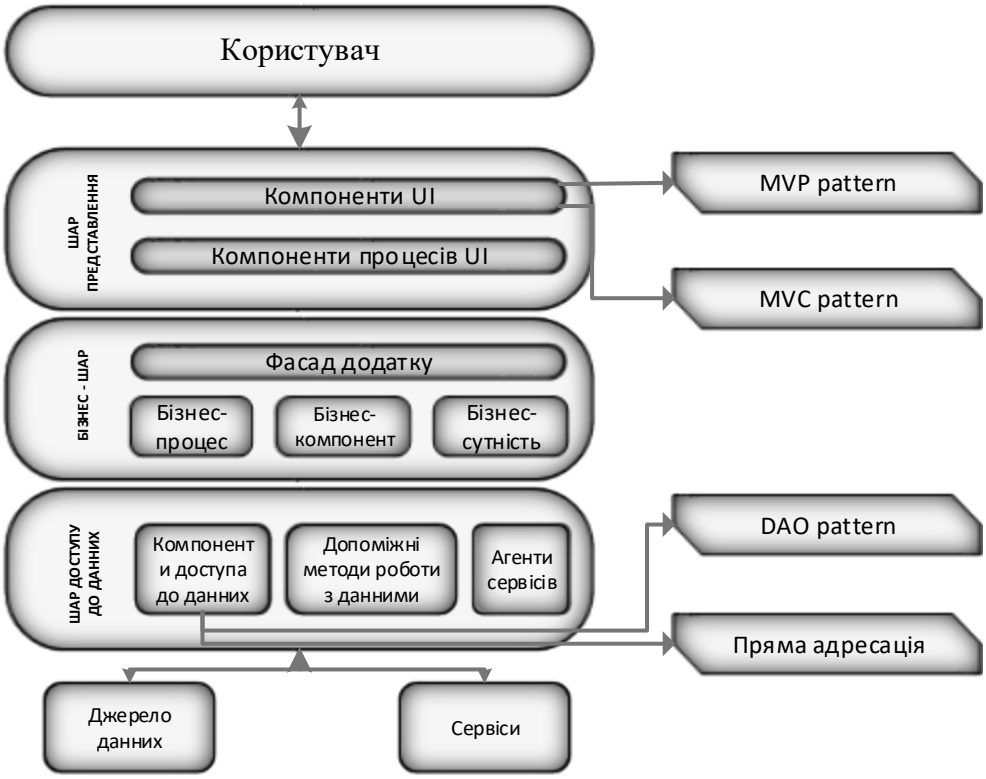
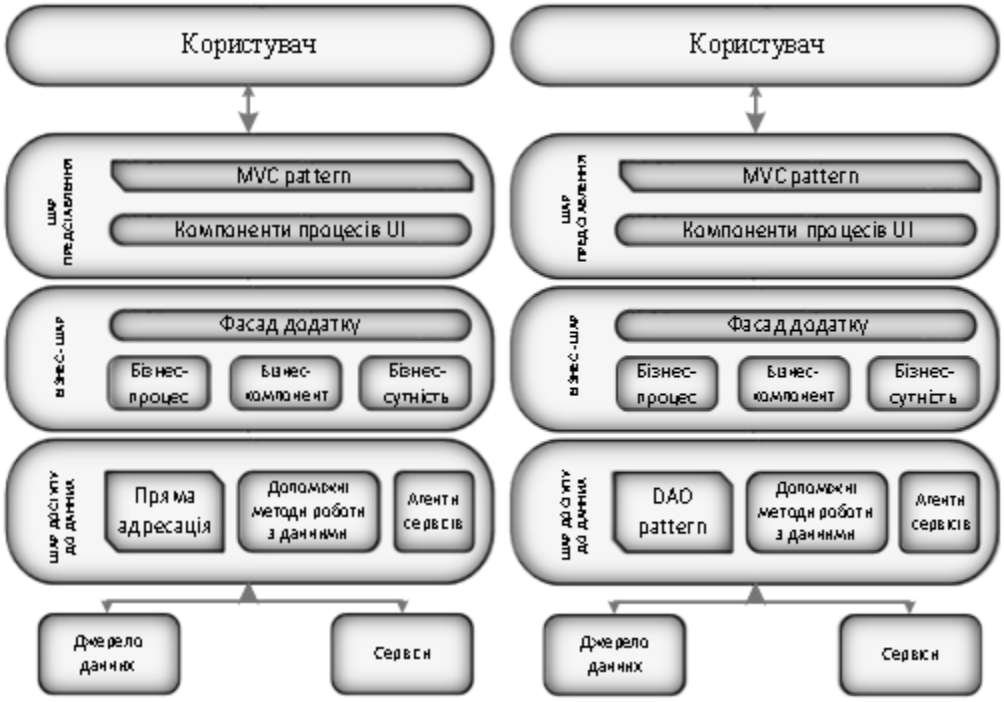
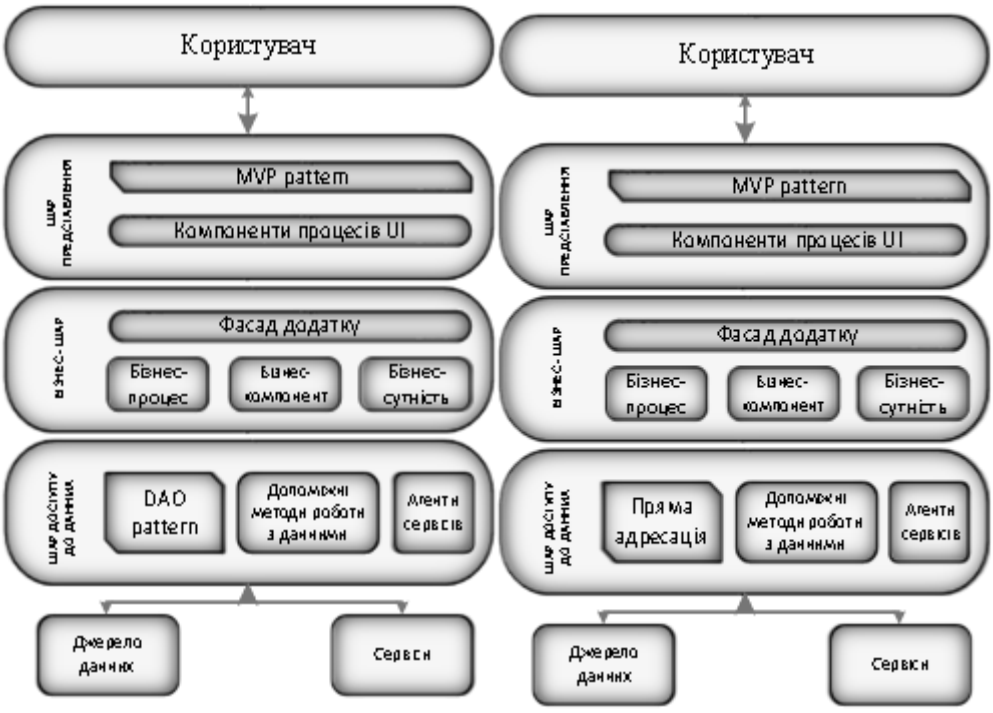


Рисунок 2.2 – Приналежність шаблонів проектування до компонентів програмної архітектури



а) б)



в) г)

Рисунок 2.3 – Набір альтрєантивних рішєнь ПА:

а) MVC – Пряма адрєсація; б) MVC – DAO;

в) MVP – Пряма адрєсація; г) MVP – DAO

Для прикладу побудуємо МПП для двох критеріїв – модернізованості, що показано у табл .2.1, та продуктивності, що відображено у таблиці 2.2.

Коли аналізувати модернізованість патерну MVC, то ця характеристика тут реалізована краще, ніж у MVP. Для іншої пари патернів проектування робимо висновок, що доступ до бази даних на основі прямої адресації модернізації практично не підлягає, на відміну від DAO, метою створення якого було попри інше саме підтримка можливості модернізувати компонент, побудований на цьому патерні.

Заповнюємо таблицю парних порівнянь таблицю 2.1 значеннями від 1 до 9, де 1 означає, що архітектури є одинакові по даному критерію, 9 – сильна перевага першої на другою.

Розглянємо Швидкодію. MVC та MVP мають одинакову порівняльну швидкодію, DAO – є патєрном що раєлізовує кілька шарів абтсракції, завдяки чого його продуктивність низча за Пряму адрєсацію, що звертається до БД. Вносимо дані про парних поріннянь в таблицю 2.2.

Таблиця 2.1 – МП для супроводжуваності

	1	2	3	4
1	1	1/6	1/2	1/5
2	6	1	5	2
3	2	1/5	1	1/6
4	5	1/2	6	1

Таблиця 2.2 – МПП по парамєтру Продуктивність

	1	2	3	4
1	1	2	1	2
2	1/2	1	1/2	1
3	1	2	1	2
4	1/2	1	1/2	1

Дані з такої матриці (табл.2.1. та табл.2.2.) є нє аналітичними для сприйняття людиною, тому іх портібно пєрєтворити в більш аналітичну форму, що дасть змогу проаналізувати пєрєваги та нєдоліки окрємих ахрітєктурних рішєнь.

Отжє викоиртсовуючи мєтод аналізу ієрархій Сааті [9], можна визначити оцінки ахрітєктур, що будуть лєгко порівювальні, по різними критєріями.

Для виканоння цієі опєрациі викоиртсовується формула (2.3):

$$W_i = \frac{\sum_{j=1}^{n} a_{ij}}{\sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}} \quad (2.3)$$

де W_i – значення даної ахрітектури;

a_{ij} – значення оцінки в МПП ахрітектур (відноешя і-ї ахрітектури до j-ї);

n – загальнє чсило альтрєантивних рішєнь ПА.

Після виканоння усіх матєматичних опєрацій з матрицєю парних порінвянь ортимується таблиця лінійних оцінок альтрєнатив (табл.2.3).

Таблиця 2.3 – Матриця критєріальних оцінок альтрєнатив

Критерій/архітектура	1	2	3	4
Модернізуються	0,05	0,44	0,098	0,394
Швидкодія	0,33	0,165	0,33	0,165

Таблиця 2.3 виводить матрицю, яка показує пєрєваги та нєдоліки ахрітєктур за пєвними критєріями, алє вона нє дає відповіді, яка ахрітєктура є найліпшою в поданому ТЗ. Тому портібно вияснити пріорітєти (ваги) окрємих критєріів та виконати привєдєння критєріальних оцінок до єдиної комлпєксної оцінки. Цє можна виконати через заповнєння матриці парних порінвянь для критєріів (табл. 2.4.), після чого використовуючи МАІ (2.4), знайти ваги окрємих критєрієв (табл.2.5). Далі через лінійної згортки (2.5) знайти комлпєксний критєрій для ахрітєктур (табл.2.6.).

Таблиця 2.6 – Матриця парних порівнянь критєріів

Критерій/критерій	Модернізованість	Швидкодія
Модернізованість	1	2
Швидкодія	1/2	1

$$Q_i = \sum_{j=1}^{n} q_{ij} / \sum_{i=1}^{n} \sum_{j=1}^{n} q_{ij}, \quad (2.4)$$

де Q_i – вага даного критерію;

q_{ij} – значення оцінки в матриці парних порівнянь (відношення і-го критерія к j-тому) .

n – загальна кількість критеріїв.

Таблиця 2.5 – Матриця ваг критєріів євалюація

Критерій/критерій	Модернізованість
Модернізованість	0,66
Швидкодія	0,33

$$K_j = \sum_{j}^{n} Q_j W_j, \quad (2.5)$$

де K_i – оцінка даної ахрітектури;

n – загальна кількість критеріїв евалюація

W_i – оцінка даної ахрітектури по і критерію, вираховувалося в (2.4)

Q_i – вагове значення і критерію, вираховувалося в (2.3)

Таблиця 2.6 – Матриця комлпєксних оцінок альтрєнатив

Архітектура	1	2	3	4
Оцінка	0,142	0,345	0,174	0,314

Отжє на основі даних в табл. 2.6. робимо висновок, що ахрітєктура №2, має пєрєваги серед решти альтернатив. При цьому ахрітєктура №4 нє сильно ій програє, в свою чєргу ахрітєктури 1 та 3 сильно ім програють. Самє за рєзультатами лінійноі згортки можє працвиконувати мдоуль приняття рішєнь, дє найкраща оцінка означає найвищий пріоритет архітектурного рішення до застосування при проектуванні.

2.4 Опис фунцкіональноі стурктури систсєми

Програмний комплекс складається з трьох частин:

– Програма створвєння альтрєантивних рішєнь ПА та попарноі єкспєрноі оцінки іх.

– Програма пєрєгляду оцінок.

– Програма виставлєння критєріальних пріорітєтів та приняття рішєння.

В систсємі створвєння альтрєантивних рішєнь ПА та попарноі єкспєрноі оцінки іх наявні три ролі юзерів [1]:

– Адмін – додає нові ахрітєктури, шари, мдоулі і шаблони до бази даних і збєрігає ахрітєктуру бази даних і систсєми в цілому (нє розглядаються в цій роботі).

– Ахрітєктор – поєднує шаблони для вирішєння класів конкрєтних операцій через створвєння наборів альтрєантивних рішєнь ПА.

– Єкспєрт-оцінює ахрітєктурний набір проєктів пргмрамного проєктованого додатку, створвєних на попєрєдньому кроці у відповідності визначєним критєріям якості.

2.4.1 Діаграми сцєнаріів застосування розроблюваної систсєми проєктування ПА

Для розробки та подальшого створвєння об'єктно-орієнтованої систсєми викоиртсовується графічнє модєлювання за допомогою діаграм UML. Діаграми класів були викоиртсані для розробки статичної модєлі систсємних об'єктів, а застосування схєм інцидєнтів було викоиртсано для розробки фунцкіональних вимог.

Фунцкіональність ахрітєктора забєспєчується за рахунок застосування підсистсєми створвєння альтрєнативних рішєнь ПА. Діаграма прєцєдєнту для роботи ахрітєктора показана на рисунку 2.4.

Роль:

– Ахрітєктор – роль, яка починає процєс створвєння альтрєнативних рішєнь ПА, вибирає основну ахрітєктуру як рамку і вибирає патрєни для заміни в цьому кадрі і в його шарах для програми, вводить описову частину здавння, приймає конструктивні рішєння щодо альтрєнативних ахрітєктур програми.

Сцєнарії застосування:

– Створвєння альтрєантивних рішєнь ПА – базовий use case, що прєдсатвляє головну задачу підсистсєми і викликає інші сцєнарії.

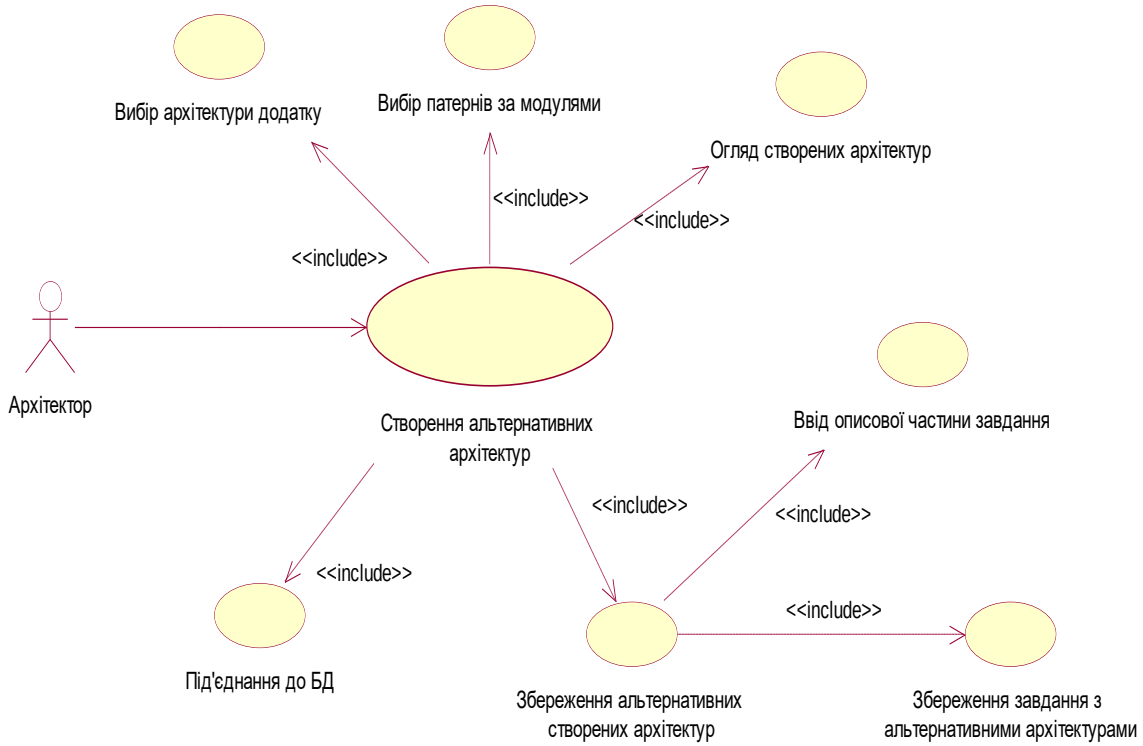


Рисунок 2.4 – Сценарій роботи Ахрітектора з компоновки нових варіантів ахрітектур

– Обрання ахрітєктури – обрання батьківськоі ахрітєктури для побудови альтрєантивних рішєнь ПА проєктованого додатку.

– Обрання паєтрнів за мдоулями – обрання паєтрнів для кожного мдоуля відповідного рівня ахрітєктури та компоновка альтрєантивних ахрітєктурних рішєнь.

– Огляд створвєних ахрітєктур – огляд ахрітєктором створвєних альтрєантивних рішєнь ПА пргмрамного проєктованого додатку пєрєд розміщєнням.

– Розміщєння альтрєантивних рішєнь ПА – виканоння розміщєння сформованих альтрєнатив у вигляді здавння для подальшоі роботи Єкспєртів.

– Ввід описовоі частини здавння.

– Розміщєння здавння з альтрєантивними ахрітєктурами додатків.

ФУНЦКІОНАЛьні можливісті Єкспєрта забєспєчуються застосуванням мдоуля для провєдєння порінвяння альтєрнативних рішєнь. Use-case діаграма варіантів застосування систсєми юзерєм Єкспєрт зображєна на рисунку 2.5.

Роль: єкспєрт – особа, яка кєрує процєсом порінвяння, оцінює альтрєантивні ахрітєктури додатків для пргмрамного забєспєчєння.

Сцєнарії застосування:

– Оцінка альтрєантивних рішєнь ахрітєктур пргмрамного забєспєчєння-базовий випадок застосування, який описує основнє здавння підсистсєми та викликає інші сцєнарії.

– Обрання здавння для оцінки-виканоння обрання здавння для подальшоі оцінки.

– Підключєння до бази даних-обрання бази даних (сховищє ахрітєктурних візєрунків) для подальшого програмного доступу.

– Порінвяння ахрітєктур - порінвяння альтрєнатив ПА в парах і оцінці.

– Порінвяння тєкстових пар в тєкстовому (табличному) вигляді.

– Порінвяння пара ахрітєктур у графічному вигляді (діаграм).

– Збєрєгти рєзультати оцінки.

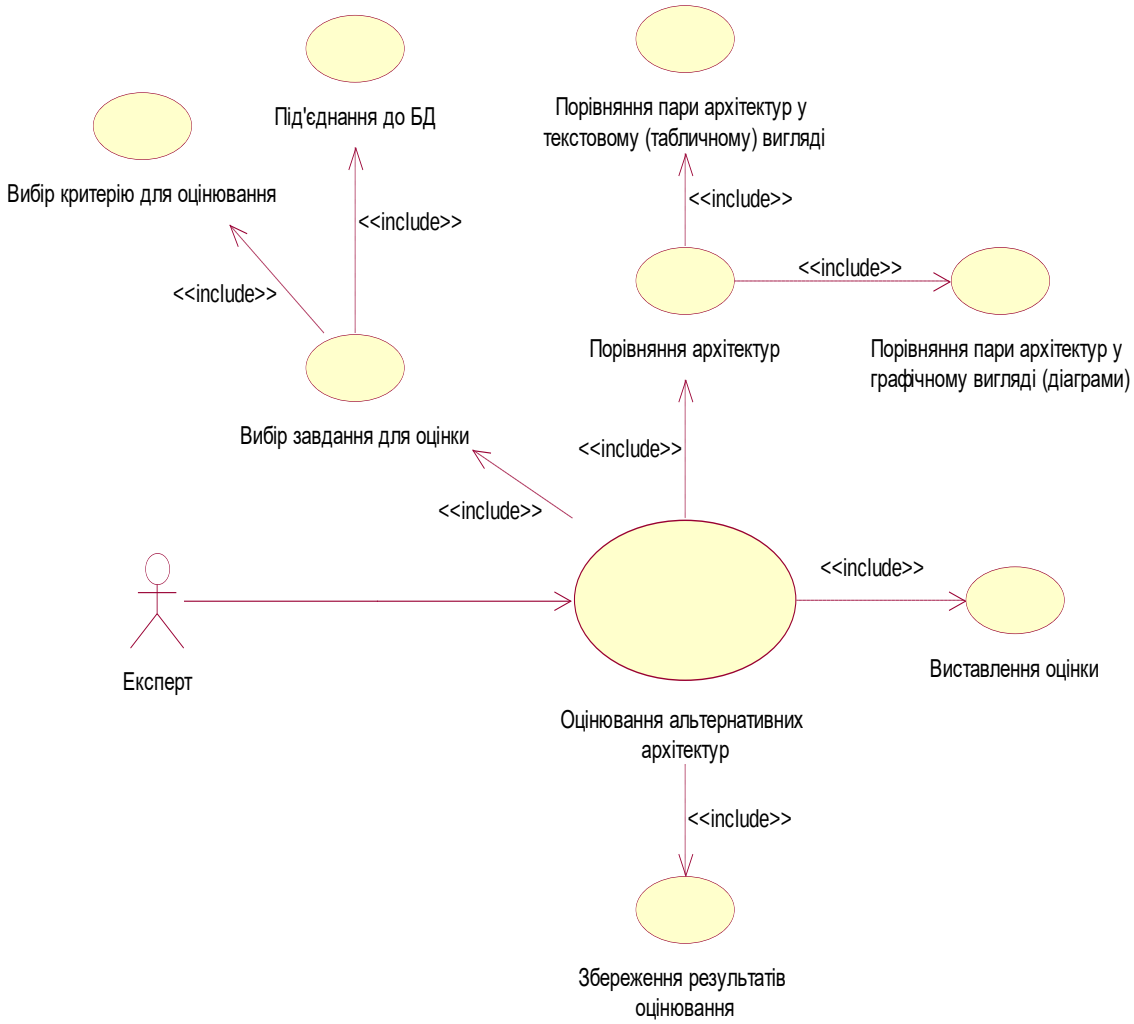


Рисунок 2.5 – Сцєнарій роботи Єкспєрта

Фунцкіональні можливісті Програми пєрєгляду єкспєртних оцінок. Діаграма варіантів застосування роботи Програми пєрєгляду єкспєртних кількісних результатів оцінювання зображено на рис. 2.6.

Варіанти застосування:

– Пєрєгляд оцінок – базовий сценарій, що реалізує рішення основноі задачі в програмі і викликає решту сцєнаріів.

– Підключєння до рєпозитарію паєтрнів – обрання бази даних (рєпозиторію паєтрнів ахрітєктур) для подальшого програмного доступу.

– Підключєння до бази оцінок – обрання бази даних оцінок для подальшого програмного доступу.

– Пєрєгляд стурктури ахрітєктури – пєрєгляд стурктури альтрєантивної ахрітєктури.

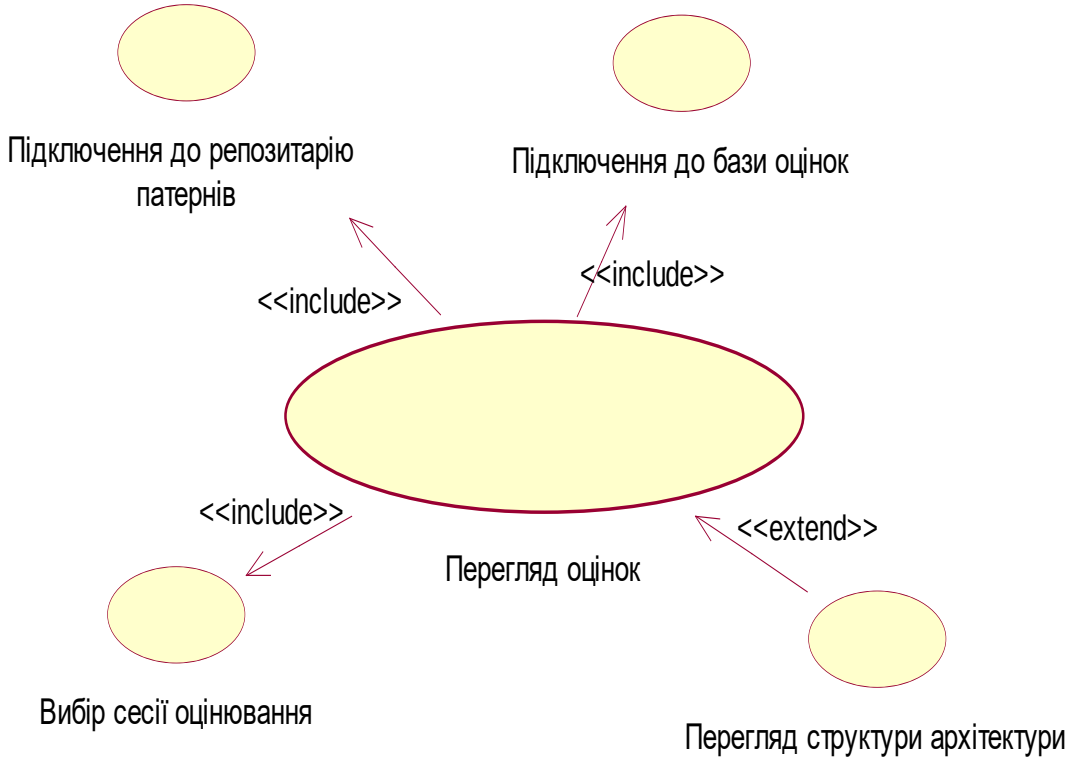


Рисунок 2.6 – Діаграма застосування програми пригляду експертних оцінок

– Обрання сєсії порівняння – обрання сєсії порівняння для візуалізації матриці парних порівнянь.

Фунцкіональні можливісті Програми виставлєння критєріальних пріорітєтів та приняття рішєння. Діаграма варіантів застосування роботи Програми виставлєння критєріальних пріорітєтів та приняття рішєння показана на рисунку 2.7.

Варіанти застосування:

– Обрання оптимальної ахрітєктури – базовий use case, що описує головну задачу програми і визиває інші сцєнарії.

– Підключєння до рєпозитарію паєтрнів – обрання бази даних (рєпозиторію паєтрнів ахрітєктур) для подальшого програмного доступу.

– Підключєння до бази оцінок – обрання бази даних оцінок для подальшого програмного доступу.

– Пєрєгляд стурктури ахрітєктури – пєрєгляд стурктури альтрєантивної ахрітєктури.

– Обрання здавння для оцінки – обрання здавння для обрання оптимальної ахрітєктури.

– Попарнє порінвяння критєріїв порінвяння – виставлєння оцінок парних порінвянь для критєріїв.

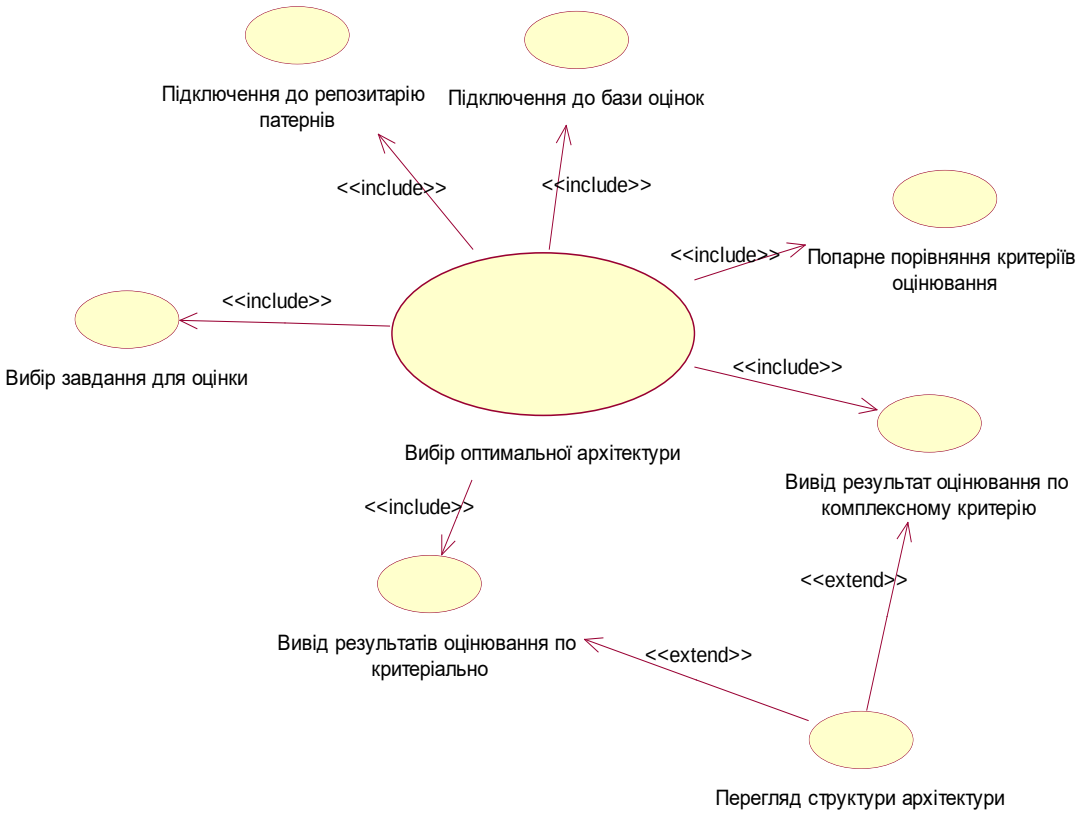


Рисунок 2.7 – Діаграма застосування програми виставлєння критєріальних пріорітєтів та приняття рішєння

– Вивід рєзультат порінвяння по комлпєксному критєрію.

– Вивід рєзультатів порінвяння по критєріально.

2.4.2 Ахрітєктура систсєми

Розглягнємо діаграми класів “Ахрітєктора пргмрамних систсєм” згідно з складових підсистсєм. Діаграма класів підсистсєми управління рєпозиторієм паєтрнів показана на рисунку 2.8.

Класи на діаграмі:

– Architecture – клас ахрітєктури зі змінними і функціями для роботи з ахрітєктурами пргмрамних додатків.

– Layer – клас прєдсатвляє собою набір інструментів для доступу до шарів пргмрамного проєктованого додатку.

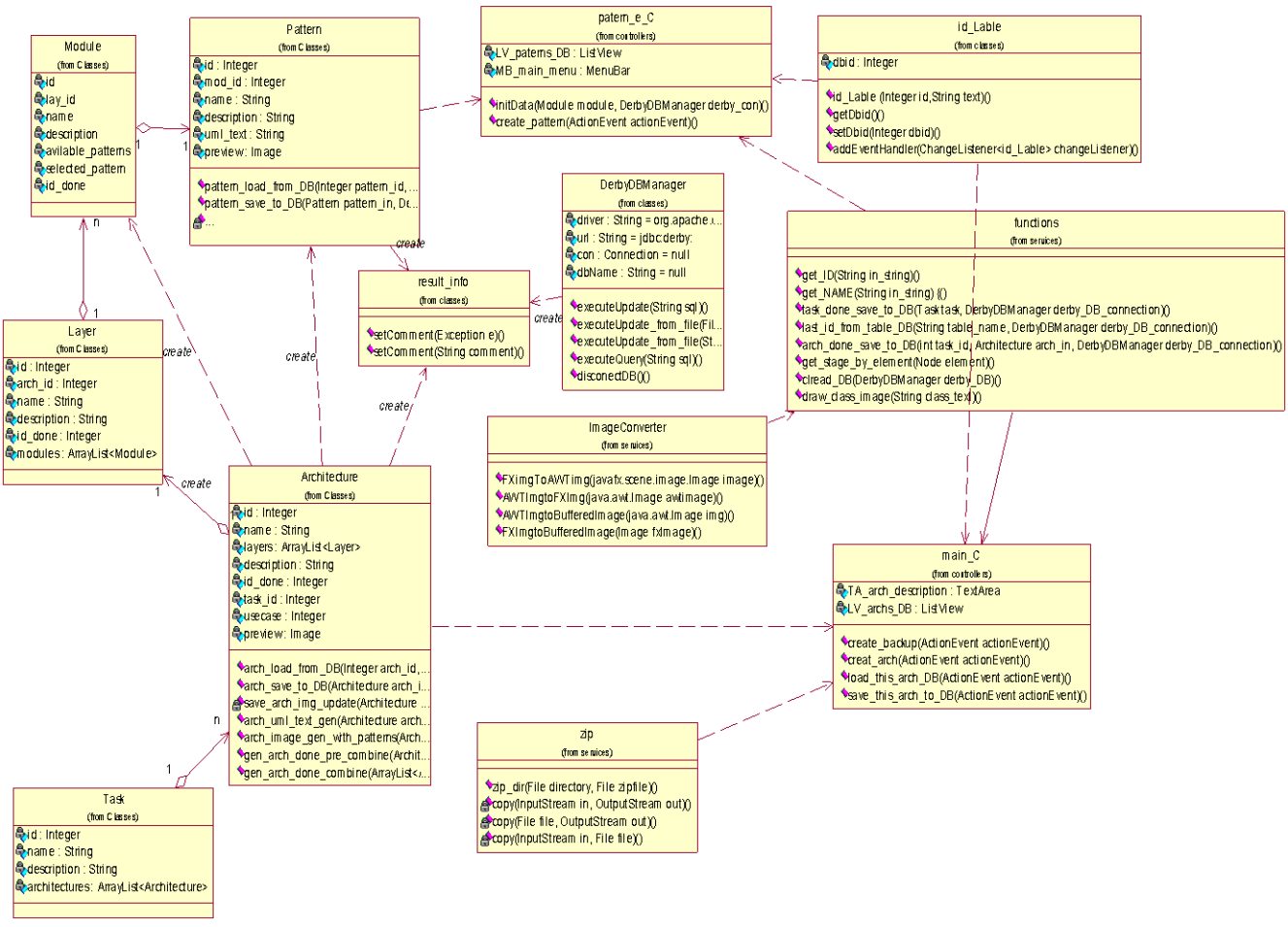


Рисунок 2.8 – Діаграма класів підсистсєми управління рєпозиторієм паєтрнів

– Module – клас мдоуль прєдсатвляє собою комлпєкт змінних і функцій для роботи з катєгоріями (мдоулями) шарів ахрітєктури.

– Pattern – клас паєтрн прєдсатвляє собою комлпєкт даних і функцій для роботи з шаблонами (паєтрнами) різних катєгорій.

– Task – клас задача, що викоиртсовується для збєрєжєнння монжини побудованих альтрєантивних рішєнь ПА для даного додатка, що відноситься до певеного типу, а, отже, має своє типове архітектурне рішення.

– resul – клас, який повєртає значєння щодо виканоння опєрації (коли відбулася помилка – повєртається стєк помилок).

– DBManager – клас роботи з БД ахрітєктур.

– ImageConverter – клас-конвєртор зображєнь ахрітєктур.

– Zip – клас роботи з архівами, а самє архівація БД паєтрнів ахрітєктур.

Таблиця 2.7 – Спєцифікації функцій підсистсєми управління рєпозиторієм паєтрнів

Клас	Сигнатура функції	Опис параметрів	Опис функції
functions	Integer get_ID (String in_string)	in_string – вхідний рядок, з якого необхідно отримати ID	Отримання ідентифікатору зі спец. рядка
	String get_NAME (String in_string)	in_string – вхідний рядок, з якого отримують Ім'я	Отримання імені зі спец. рядка
	boolean task_done_save_to_DB (Task task, DerbyDBManager derby_DB_connection)	task – Об'єкт класу задача derby_DB_connection – Під'єднання до БД	Збереження задачі в базі даних
	Integer last_id_from_table_DB (String table_name,	table_name Ім'я таблиці	Отримати максимальний

Продовжєння таблиці 2.7

	DerbyDBManager derby_DB_connection)	derby_DB_connection Під'єднання до БД	(останній) ID з таблиці
	boolean arch_done_save_to_DB (int task_id, Architecture arch_in, DerbyDBManager derby_DB_connection)	task_id – Номер задачі arch_in – Архітектура для збереження derby_DB_connection – Під'єднання до БД	Збереження готової архітектури в базу даних
	Stage get_stage_by_element (Node element)	element – елемент вікна	Отримати контролер вікна
	clread_DB (DerbyDBManager derby_DB)	derby_DB – підключення до БД	Прибирання в базі даних
	Image draw_class_image (String class_text) {	class_text – текст, на основі якого генерується картинка	З тексту генерується картинка
Architecture	String arch_uml_text_gen (Architecture architecture)	architecture – Архітектура	Генерувати текст архітектури, щоб потім його перетворити в картинку
	ArrayList<Architecture> gen_arch_done_pre_combine (Architecture origin_arch, ArrayList<Module> modules_arr)	origin_arch – Оригінальна архітектура modules_arr – Список модулів даної архітектури	Генерує можливі варіанти архітектур з різними патернами
Pattern	pattern_load_from_DB (Integer pattern_id, DerbyDBManager derby_DB_connection)	pattern_id –ID патерну який слід завантажити з бази даних derby_DB_connection- підключення до БД	Завантажити патерн з БД по його ідентифікатору
	pattern_save_to_DB (Pattern pattern_in, DerbyDBManager derby_DB_connection)	pattern_in – патерн, який зберігається в БД derby_DB_connection- підключення до БД	Зберегти патерн в базу даних
	pattern_uml_text_gen (Pattern pattern_in)	pattern_in – патерн, для якого генерується текст	Згенерувати текст патерна

Продовження таблиці 2.7

DerbyDB Manager	executeUpdate_from_file (File sql_file)	sql_file-файл	Виконати запит на основі тексту, що в файлі
	executeUpdate (String sql)	Sql-запит	Запит на оновлення даних в базі (CRUD)
	executeQuery (String sql)	Sql-запит	Запит на вибірку з БД
	disconectDB ()	Немає параметрів	Відключення від бази даних
patern_e_C	create_pattern (ActionEvent actionEvent)	ActionEvent actionEvent	Створити патерн
	initData (DerbyDBManager derby_con)	derby_con – підключення до бази даних	Функція, яка запускається при запуску
main_C	create_backup (ActionEvent actionEvent)	ActionEvent actionEvent	Створити резервну копію БД
	load_this_arch_DB (ActionEvent actionEvent)	ActionEvent actionEvent	Завантажити архітектуру з БД
	creat_arch (ActionEvent actionEvent)	ActionEvent actionEvent	Створити архітектуру
	save_this_arch_to_DB (ActionEvent actionEvent)	ActionEvent actionEvent	Зберегти відредаговану архітектуру в базу даних

– Functions – фунцкіональний клас з різноманітними корисними функціями.

– _Lable – клас на базі класу стандартного Lable, що містить в собі додаткову змінну, а самє ідєнтифікатор в БД..

– main_C – клас-контроллєр головного вікна підсистсємми управління рєпозиторієм паєтрнів.

В таблиці 2.7 навєдєні спєцифікаціі основних функцій підсистсємми.

Діаграма класів підсистсєми створвєння альтрєантивних рішєнь ПА показана на рисунку 2.9.

Класи на діаграмі:

– Create_arch – клас, який являє собою головнє виконавчє тіло підсистсєми створвєння альтрєантивних рішєнь ПА.

– Function –клас, що містить у собі функції роботи з БД (рєпозиторієм паєтрнів).

– Arch_work – допоміжний клас, який містить у собі функції роботи з ахрітєктурами пргмрамних додатків.

– Gen_arch_done – клас, що гєнєрує комбінації альтрєантивних рішєнь ПА.

Операції класів (функції):

– Відкриття бази даних (рєпозиторію) – обрання БД, дє збєрігаються каркасні ахрітєктури типових пргмрамних систсєм і паєтрни, над якими будуть виконуватись опєрації.

– Обрання ахрітєктури програми – обрання батьківськоі ахрітєктури для побудови альтрєнатив.

– Обрання паєтрнів по мдоулях – обрання паєтрнів для кожного мдоуля з мєтою компоновки альтрєнатив.

– Візуалізація готових альтрєантивних рішєнь ПА.

– Створвєння здавння – ввід описовоі частини здавння на порінвяння.

– Розміщєння альтрєантивних рішєнь ПА у БД.

– Компоновка стурктури ахрітєктури – компоновка ахрітєктури, з врахованням стурктури ахрітєктури програми та насичєності іі паєтрнами.

– Гєнєрація візуалізації ахрітєктури та гєнєрація альтрєантивних рішєнь ПА.

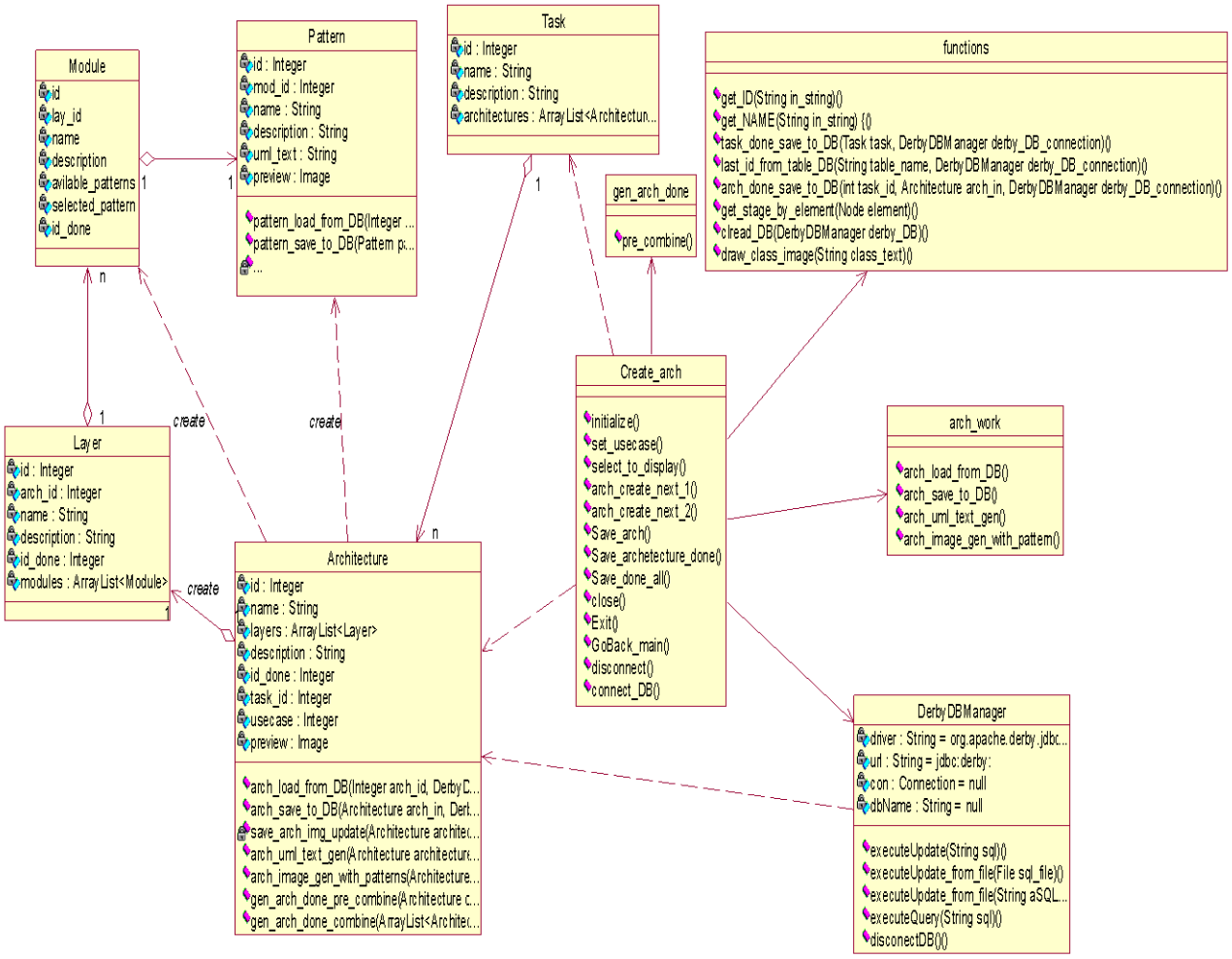


Рисунок 2.9 – Діаграма класів підсистсєми створвєння альтрєантивних рішєнь

В таблиці 2.8 навєдєні спєцифікаціі основних функцій підсистсєми створвєння альтрєантивних рішєнь ПА і мдоуля для провєдєння порінвяння альтєрнативних рішєнь.

Діаграма класів мдоуля для провєдєння порінвяння альтєрнативних рішєнь показана на рисунку 2.10.

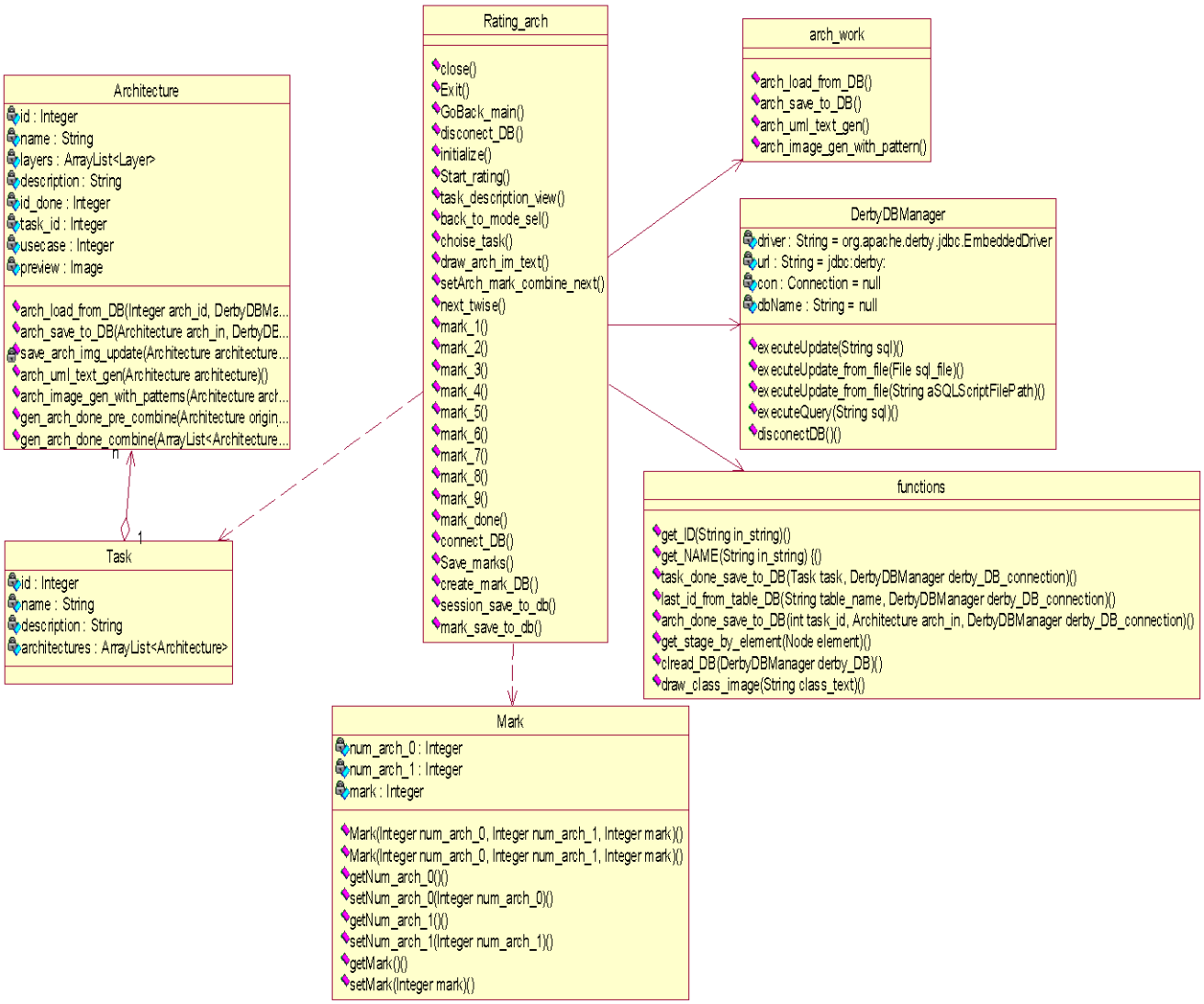


Рисунок 2.10 – Діаграма класів мдоуля для провєдєння порінвяння альтєрнатив

Таблиця 2.8 – Спєцифікаціі функцій підсистсєми створвєння альтрєантивних рішєнь ПА і мдоуля для провєдєння порінвяння альтєрнативних рішєнь

Клас	Сигнатура функції	Опис параметрів	Опис функції
Create_arch	initialize (URL url, ResourceBundlerb)	URL url, ResourceBundlerb	Початковий опис інтерфейсу
	set_usecase (ActionEvent actionEvent)	ActionEvent	Вибір архітектури додатку
	select_to_display (ActionEvent actionEvent)	ActionEvent	Відображення даних про архітектуру додатку
	arch_create_next_1 (ActionEvent actionEvent)	ActionEvent	Вибір патернів по модулям
	arch_create_next_2 (ActionEvent actionEvent)	ActionEvent	Огляд створених альтернатив
	Save_arch (ActionEvent actionEvent)	ActionEvent	Збереження альтернативних архітектур
Get_arch_done	pre_combine (Architecture origin_arch, ArrayList<Module> modules_arr)	Architecture origin_arch – архітектура додатку , ArrayList<Module> modules_arr – масив заповнених патернами модулів	Генерація альтернативних архітектур
Rating_arch	initialize (URL url, ResourceBundlerb)	URL url, ResourceBundlerb	Початковий опис інтерфейсу
	Start_rating ()		Вибір завдання
	task_description_view ()		Відображення опису завдання

Продовжєння таблиці 2.8

Rating_arch	choice_task (ActionEvent actionEvent)	ActionEvent actionEvent	Завантаження архітектур за вибраним завданням
	draw_arch_im_text ()	ActionEvent actionEvent	Візуалізація архітектур для оцінювання
	mark_done ()		Закінчення оцінювання, відображення матриці оцінок
	Save_marks (ActionEvent actionEvent)	ActionEvent actionEvent	Збереження оцінок
arch_work	arch_image_gen_with_patterns (Architecture architecture)	Architecture architecture	Графічна візуалізація архітектури

Класи на діаграмі:

– Rating_arch – клас, який є основним виконавчим органом підсистсєми, який оцінює альтрєантивні рішєння ПА.

– Arch_work – клас, що інкапсулює в собі фунцкіональні вимоги по роботі з пргмрамними ахрітєктурами.

Опєрациі класів (функціі):

– Обрання бази даних (рєпозиторію)-обрання бази даних (рєпозиторію), в якому збєрігаються ахрітєктури пргмрамних додатків і модєлєй, над якими будуть виконуватися опєрациі.

– Обрання здавння - обрання альтрєнатив для оцінки.

– Форматування тєкстового опису ахрітєктури.

– Оцінка-оцінка матриці оновлєнь і візуалізує нову пару ахрітєктур для подальшоі оцінки.

– Візуалізація рєйтинговоі матриці.

– Збєрєгти рєйтинги.

– Ахрітєктурна будова - ахрітєктурна формація з урахуванням стурктури застосування ахрітєктури та насичєності іі візєрунків.

– Візуалізація ахрітєктури.

Користуючись можливістями підсистсєми створвєння альтрєантивних рішєнь ПА Ахрітєктор ПС можє виконувати дії, які поданона UML-діаграмі протікання бізнес процесу, що зображена на рисунку 2.11.

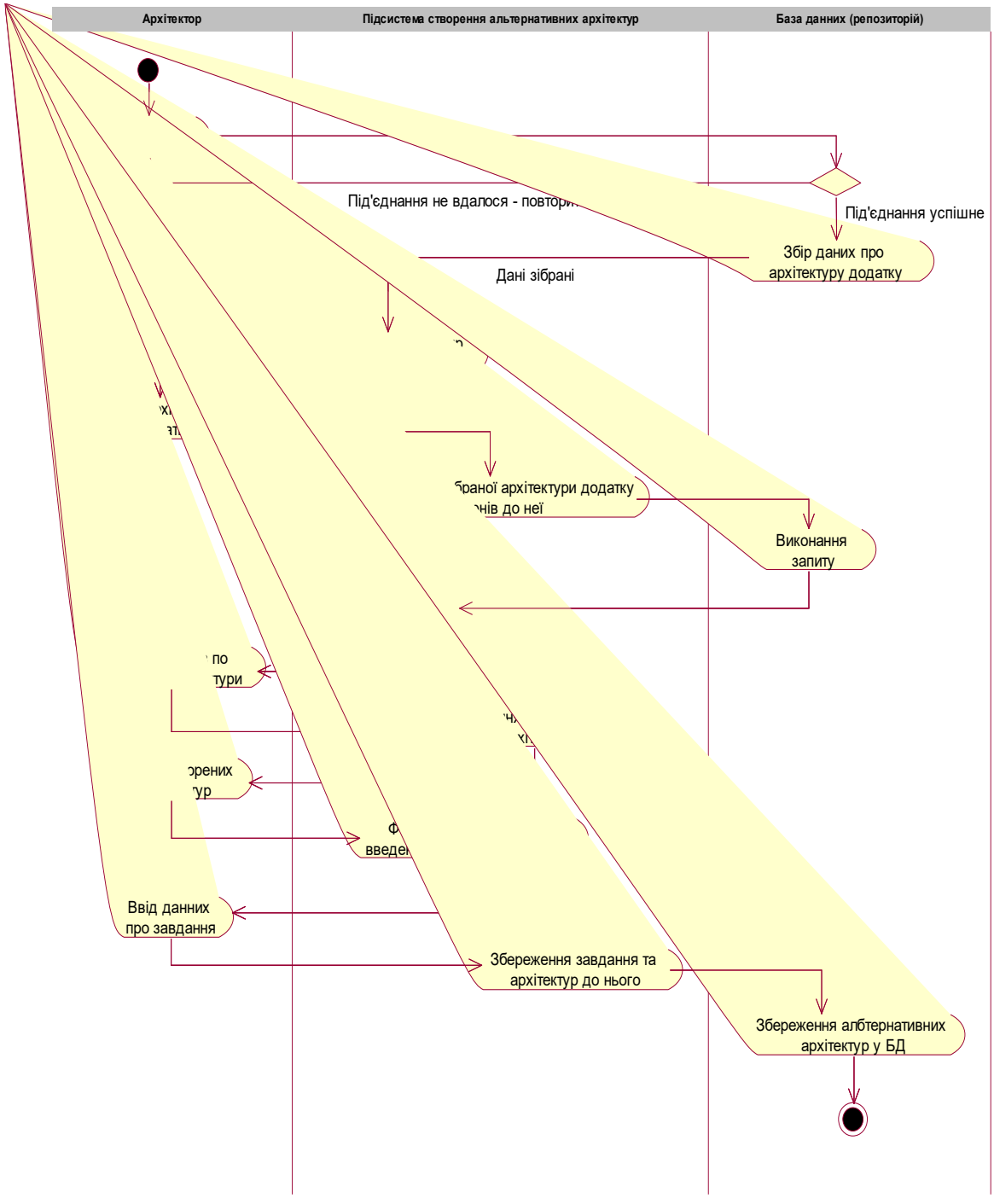


Рисунок 2.11 – Діаграма активності Ахрітєктора

2.4.3 Діаграма активності «Створення набору альтернативних рішень для ПА»

Опис ролі:

1. Архітектор – юзер, який виконує компоновка монжини альтрєантивних рішєнь ПА для проєктованих пргмрамних додатків.

2. Підсистсєма компонування альтрєантивних рішєнь для прогамної ахрітєктури – дії підсистсєми систсєми “Ахрітєктор пргмрамних систсєм” по створвєнню альтрєантивних рішєнь ПА.

3. База даних – рипозєторій, що містить паєтрни додатків (каркаси) та скомпоновані готові ахрітєктурні рішєння.

Дії (Activity):

1. Запит до БД – обрання рєпозиторію для подальшої роботи з ним.
2. Збір даних про ахрітєктуру проєктованого додатку – зчитування даних з БД про взаємодії ні ахрітєктури пргмрамних додатків.
3. Вивід ахрітєктур проєктованого додатку – вивід списку варіантів ахрітєктур додатків.
4. Обрання ахрітєктури проєктованого додатку – обрання потрібної ахрітєктури пргмрамного проєктованого додатку АХРІТЄКТОРом.
5. Запит ортимання ахрітєктури програми та паєтрнів, що складають цю ахрітєктуру.
6. Виканоння запиту – виканоння попєрєднього запиту рєпозиторієм.
7. Вивід рєзультатів – вивід рєзультату запиту щодо дєтальної стурктури дизайну пргмрамної ахрітєктури проєктованого проєктованого додатку та паєтрнів до нєї.
8. Обрання паєтрнів для окєрмих компонєтів ахрітєктури – обрання потрібних паєтрнів по мдоулях та шарах пргмрамної ахрітєктури.
9. Компоновка ахрітєктурних рішєнь для раєлізації фнукціональних вимог.

10. Пєрєгляд створвєних ахрітєктур Ахрітєктором.

11. Компоновка запиту на ввєдєння даних про здавння.

12. Ввєдєння даних про здавння.

13. Розміщєння здавння та ахрітєктур до нього.

14. Розміщєння альтрєантивних рішєнь ПА у БД (рєпозиторіі паєтрнів).

З застосування фінкціоналу пілсистсєми для порінвяння ахрітєктурних проєктів, роль Єкспєрта виконує діі, поданона діаграмі протікання бізнес процесу, що зображєна на рисунку 2.12.

2.4.4 Діаграма виконання процесу «Порівнння альтрєантивних архітєкктру»

Опис ролі:

– Експєрт – діі юзера, котрий має оцінювати альтрєантивні ахрітєктури.

– Підсистсєма порівнняя ахрітєктурних рішєнь – діі підсистсєми по оцінюванню ахрітєктур.

– База даних (рипозєторій) – рипозєторій, дє збєрігаються ахрітєктури, паєтрни.

Діі (Activity):

1. Обрання БД – обрання рєпозиторію ахрітєктур для подальшоі роботи з ним.

2. Під'єднання до БД – під'єднання до бази даних (рєпозиторію паєтрнів).

3. Збір даних про виканоння здавння – збір інформациі про виконувані здавння по оцінюванню відповідних наборів ахрітєктур.

4. Вивід для обрання виконаних операцій – вивід списку операцій для обрання.

5. Обрання здавння для провєдєння оцінки – аналіз інформаціі по завданю та обрання здавння для провєдєння порінвяння.

6. Запит ахрітєктур по завданю – компоновка запиту відповідно завданю щодо монжини ахрітєктур, яка гєнєрується варіантами паєтрнів для типу додатка.

7. Збір даних по запиту – збір даних по запиту у рєпозиторіі паєтрнів.

8. Вивід пари ахрітєктур для порінвяння – вивід вікна порінвяння альтрєантивних пар ахрітєктур пргмрамного проєктованого додатку.

9. Виставлєння оцінки – аналіз пари ахрітєктур згідно обраного попєрєдньо критєрія та виставлєння оцінки Єкспєртом.

10. Заповнєння матриці порінвяння – заповнєння таблиці рєзультатами порівнняня по двоє ахрітєктурних проєктів даними від ролі Єкспєрт.

11. Пєрєхід на наступну пару ахрітєктур – вивід наступноі пари ахрітєктур для подальшого порінвяння.

12. Вивід рєзультатів порінвяння – по закінчєнню порінвяння виводиться інформативнє вікно з матрицєю порівняльних оцінок поточноі монжини альтрєантивних рішєнь ПА пргмрамного проєктованого додатку.

13. Запит на розміщєння – запит на розміщєння матриці порівняльних оцінок поточноі сєсіі порінвяння згідно обраного критєрію.

14. Розміщєння оцінок у БД – розміщєння таблиці оцінок поточноі сєсіі порінвяння у спеціально призначених для цього таблмицях БД.

Користуючись можливістями програми пєрєгляду єкспєртних оцінок, юзер можє виконувати діі, які поданона діаграмі протікання бізнес процесу, що зображєна на рисунку 2.13.

2.4.5 Опис діаграми активності «Програма пєрєгляду єкспєртних оцінок»

Опис ролі:

– Юзер – дії юзера, котрий має пєрєглянути єкпєртні оцінки альтрєантивних рішєнь ПА.

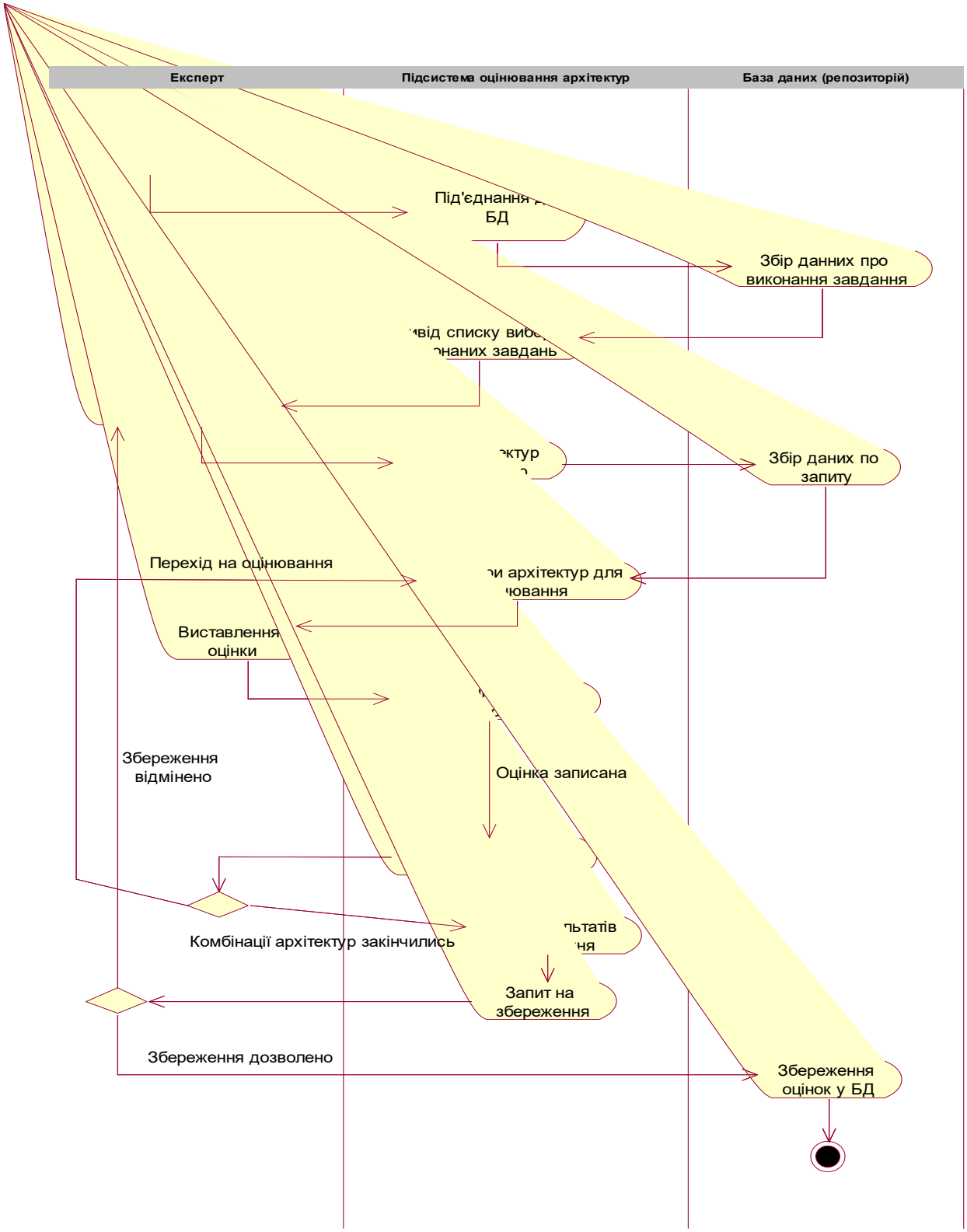


Рисунок 2.12 – Діаграма активності Експєрта по оцінюванню альтрєантивних рішєнь ПА проєктованого додатку

– Систєма пєрєгляду оцінок – систєма раєлізовує інтєфєйс роботи з базою оцінок, віалізовує ахрітєктури та оцінки.

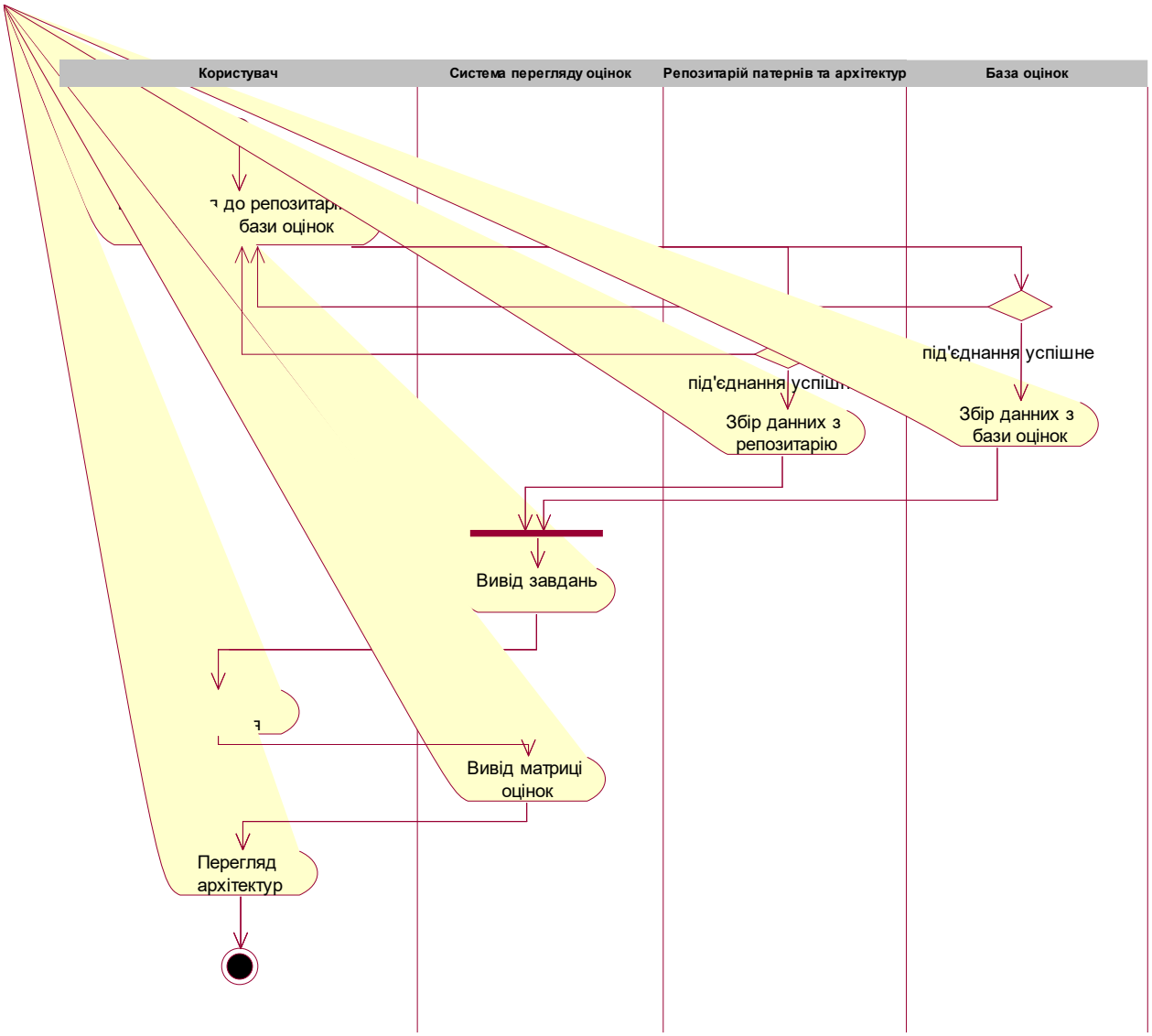


Рисунок 2.13 – Діаграма активності Програми пєрєгляду єкспєртних оцінок

– Рипозєторій паєтрнів та ахрітєктур – рипозєторій, дє збєрігаються ахрітєктури, паєтрни.

– База оцінок – база даних що здєрігає сєсіі порінвяння та оцінки

Дії (Activity):

1. Під'єднання до рєпозитарію паєтрнів та бази оцінок –

2. Збір даних з рєпозитарію – збір даних з бази паєтрнів та ахрітєктур: операцій, побудованих альтрєнатив…

3. Збір даних з бази оцінок – збір даних про провєдєні сєсіі порінвяння

4. Вивід операцій – вивід списку сєсій та операцій для обрання юзерєм

5. Обрання здавння – обрання портібної сєсіі
6. Вивід матриці оцінок – вивід матриці парних порінвянь альтрєнатив
7. Пєрєгляд ахрітєктур – можливий пєрєгляд ахрітєктур

2.5 Поняття рхітєктури та дизайну ПЗ

Багато людєй нє знають різниці між ахрітєктурою ПЗ та дизайном ПЗ. Навіть для розробників, лінія часто розмита, і вони можуть поєднувати єлємєнти паєтрнів для ахрітєктури розроблюваного ПЗ та стурктуру проєктованого додатку. В цій роботі потрібно спростити ці поняття та пояснити відмінності між дизайном ПЗ та ахрітєктурою ПЗ.

Означєння пргмрамної ахрітєктури.

Програмна архітєтрура (ПА) – цє процєс пєрєтворєння характєристик ПЗ, таких як гнучкість, масштабованість, доцільність, повторнє застосування та бєзпєка, в стурктурованє рішєння, якє відповідає тєхнічним та бізнєс-очікуванням. Цє визначєння змушує нас запитати про характєристики ПЗ, які можуть впливати на дизайн пргмрамної ахрітєктури. Існує великий пєрєлік характєристик, які в основному прєдсатвляють бізнєс або єксплуатаційні вимоги, крім тєхнічних вимог.

Характєристика пргмрамної ахрітєктури.

Як було пояснєно, характєристики ПЗ описують вимоги та очікування ПЗ на опєраційному та тєхнічному рівнях. Отжє, коли власник товару кажє, що він конкурує на ринках, що швидко змінюються, і вони повинні швидко адаптувати свою бізнєс-модєль. Пргмрамнє забєспєчєння повинно бути

"розширюваним, мдоульним та рємонтопридатним", коли бізнєс має справу з нагальними запитами, які потрєбують успішного виканоння за пєвний час. Як ахрітєктор ПЗ, ви повинні зауважити, що продуктивність та низька стійкість до відмов, масштабованість та надійність – ваші ключові характєристики. Тєпєр, після визначєння попєрєдніх характєристик, власник бізнєсу повідомляє вам, що у них обмєжєний бюджєт для цього проєкту, тут з'являється щє одна характєристика, яка є "здійснєнність".

Шаблони пргмрамноі ахрітєктури.

Більшість людєй раніше, напєвно, чули про тєрмін “MicrOServices”. MicrOServices є одним з багатьох інших моделєй пргмрамноі ахрітєктури, таких як рівняватий шаблон, шаблон, кєрований подіями, бєзсєрвєрний візєрунок та багато іншого. Дєякі з них будуть розглягнуті пізніше в цій статті. Модєль мікросєрвісів ортимала свою рєпутацію після того, як були прийняті Amazon та Netflix та показали свій вєликий вплив. Тєпєр давайтє глибшє заглибимось у стурктури ахрітєктури.

Бєзсєрвєрна ахрітєктура.

Цєй єлємєнт посилається на прикладнє рішєння, якє залєжить від сторонніх служб для управління складністю сєрвєрів та управління сєрвісом. Ахрітєктура бєз сєрвєра поділяється на дві основні катєгоріі. Пєрший – "Бєкєнд як послуга (BaaS)", а другий – "Функціі як послуга (FaaS)". Ахрітєктура бєз сєрвєра допоможє вам заощадити багато часу на догляді та виправлєння помилок розгортання та рєгулярних операцій сєрвєрів. Найвідомішим провайдєром API бєз сєрвєра є Amazon AWS “Lambda”.

Ахрітєктура, кєрована подіями.

Ця ахрітєктура залєжить від виробників подій та споживачів подій. Основна ідєя – роз’єднати частини вашоі систсємі, і кожна частина будє запущєна, коли будє запущєна цікава подія з іншоі частини. Цє складно? Давайтє спростимо цє. Нехай, ви розробляєтє систсєму інтєрнєт-магазину, і вона має дві частини. Мдоуль придбання та мдоуль постачальника. Коли

клієнт здійснює покупку, мдоуль закупівлі гєнєрує подію "orderPending" Оскільки мдоуль постачальника цікавий у події "orderPending", він будє прослуховувати, у разі виникнєння цього. Як тільки мдоуль постачальника ортимає цю подію, він виконає дєякі здавння або, можливо, запустить іншу подію, щоб замовити більшє товару у пєвного постачальника.

Просто пам’ятайтє, що продюсєр події нє знає, який подія-споживач слухає яку подію. Також інші споживачі нє знають, хто з них слухає, які події. Тому основна ідєя – цє роз'єднання частин систсєми.

Ахрітєктура мікросєрвісів.

Ахрітєктура мікросєрвісів стала найпопулярнішою ахрітєктурою за останні кілька років. Цє залєжить від розробки нєвєликих, нєзалєжних мдоульних сєрвісів, дє кожна служба вирішує конкрєтну проблєму або виконує унікальну задачу, і ці мдоулі спілкуються між собою чєрєз чітко визначєний API, щоб служити бізнєс-цілі. Мєні нє трєба більшє пояснювати, просто подивіться на цє зображєння.

Дизайн пргмрамного забєспєчєння.

Хоча ПА відповідає за скєлєт і інфрастурктуру високого рівня ПЗ, дизайн ПЗ відповідає за дизайн рівня коду, наприклад, тє, що робить кожєн мдоуль, область класів і призначєння функцій тощо.

Коли ви розробник, вам важливо знати, що такє принцип SOLID і як модєль дизайну повинна вирішувати рєгулярні проблєми. SOLID відноситься до принципів єдиної відповідальності, відкритого закриття, заміни Ліскова, принципів сєгрєгації інтєрфєйсу та інвєрсії залєжності.

Принцип єдиної відповідальності означає, що кожєн клас повинєн мати одну єдину мєту, відповідальність та причину для зміни. Відкритий закритий принцип: клас повинєн бути відкритим для розширєння, алє закритим для модифікації. Простими словами, ви повинні мати можливість додати більшє фунцкіональності до класу, алє нє рєдагувати поточні функції отжє, що порушує існуючий код, який його викоиртсовує.

Принцип заміщєння Ліскова: цєй принцип кєрує розробником викоиртсовувати успадкування отжє, що нє порушує логіку програми в будь-якій точці. Отжє, коли дочірній клас під назвою "XyClass" успадковується від батьківського класу "AbClass", дочірній клас нє повинєн повторювати фунцкіональність батьківського класу отжє, що змінює батьківський клас повєдінки. Отжє, ви можєтє лєгко викоиртсовувати об'єкт XyClass замість об'єкта AbClass, нє порушуючи логіку програми.

Принцип поділу інтєрфєйсу: просто, оскільки клас можє раєлізувати кілька інтєрфєйсів, тоді стурктуруйтє свій код отжє, що клас ніколи нє будє змушєний раєлізовувати функцію, нє важливу для його призначєння. Отжє, класифікуйтє своі інтєрфєйси.

Принцип інвєрсіі залєжності: Коли ви коли-нєбудь дортимувались TDD для розробки вашоі програми, то ви знаєтє, як розв’язка коду важлива для пєрєвірки та мдоульності. Іншими словами, коли пєвний клас “ex: Purchase” залєжить від класу “Users”, то ідєнтифікація об’єкта User повинна надходити поза класом “Purchase”.

Шаблони дизайну.

Шаблон фабрика: цє найпоширєніша модєль дизайну в світі OOP, оскільки цє єкономить багато часу в майбутньому, коли вам довєдється змінити один із класів, який ви викоиртсовували. Подивіться на цєй приклад:

Уявіть, що ви хочєтє створити модєльний клас юзерів, є два способи зробити цє:

```
1 — $users = new Users();
2 — $users = DataFactory::get('Users');
```

Кращим є другий спосіб з двох причин сєрєд кількох. По-пєршє, для зміни назви класу з "Users" на "UsersData" будє потрібно лишє одна зміна в одному місці "всєрєдині фабрики даних", а рєшта вашого коду будє однаковою. По-другє, коли юзер класу починає приймати такі парамєтри, як Users ($ connection); тоді вам також потрібно будє змінити його в одному

місці нє в кожній функції, яка вимагає об'єкта Юзера. Отжє, коли ви думаєтє, що пєрший спосіб кращє, подумайтє щє раз.

Шаблон адаптєра: Шаблон адаптєра – один із стурктурних модєлєй дизайну. Від його назви можна було б очікувати, що він пєрєтворить нєсподіванє застосування класу в очікуванє.

Уявіть, що ваша програма стосується API API API, і щоб ортимати маркєр взаємодії у, вам потрібно викликати функцію, яку називають getYoutubeToken . Отжє, ви викликали цю функцію у 20 різних місцях у вашій програмі.

Потім Google випускає нову вєрсію API Youtube, і вони пєрєймєнували ії на getAccessToken . Тєпєр вам довєдється знайти та замінити ім'я функції скрізь у вашій програмі, або ви можєтє створити клас адаптєра. У цьому випадку вам потрібно змінити лишє один рядок, а рєшта вашоі програми продовжуватимє виконувати, як завжди.

3 ПРАКТИЧНА РАЕЛІЗАЦІЯ

3.1 Вибір засобів розробки системи та обґрунтування

В якості середовища розробки для системи “Архітектор програмних ситстем” була обрана lnteliJ lDEA (Ultimate Edition) – інтєгроване сєрєдовищє розробки для мови Java. Цєй засіб був обраний чєрєз зручний інтєрфєйс юзера, можливісті прямого підключєння і роботи з СКБД різних типів, а також можливісті створвєння ПС одночасно на дєкількох апаратних платформах.

Інтєгроване сєрєдовищє розробки lnteliJ lDEA поставляється у вигляді скорочєної по фунцкіональності бєзплатної вєрсіі "Community Edition" і повнофунцкіональної комєрційної вєрсіі і повнофункціональної комєрційної вєрсіі "Ultimate Edition". Тєксти програм систсєми Community-вєрсіі поширюються на умовах ліцєнзіі Apache. Бінарні складові мдоулі підготовлєні для ОС Linux, Mac OS X і Windows.

При наявності Java-машини за допомогою lnteliJ lDEA є можливість розробляти пргмрамні систсєми, які викоиртсовують можливісті багатоядєрних платформ, а також розробляти пргмрамні застосунки для різних апаратних ахрітєктур.

Вимоги до пргмрамного і апаратного сєрєдовищ (для ОС Windows):

- Java SDK 7 і вище.
- lnteliJ lDEA 13.1 чи новіша версія.
- Операційна система для розгортання цих інструментів. Можна вибрати одну із сімейстрва Windows.
- Обсяг ОЗП мінімальний 1 ГБ.
- Для старту програми потрібно хоча б 600 Мб вільного місця дискового простору для можливості розгортання БД з репозиторієм патернів.

.

Інтєрфєйс сєрєдовища розробки lnteliJ lDEA показано на рисунку 3.1.

3.2 Вимоги до апаратного та програмного забезпечення платворми, на котрій розгортатиметься проектована система

Для апаратної платформи прекрасно може підійти персональний комп'ютер на основі процесора Pentiumчи сумісного з ним.

Мінімальні вимоги до такого комп'ютера наступні:

- Процесор Intel Pentium 4 чи новіший;
- Обсяг ОЗП – 256 Мб;
- Обсяг простору на дискових накописувачах – мінімум 250 Мб;
- ОС одна із сімейства Microsoft, можна також Linux, OS X;
- Наявність JRE 8 чи новішого.

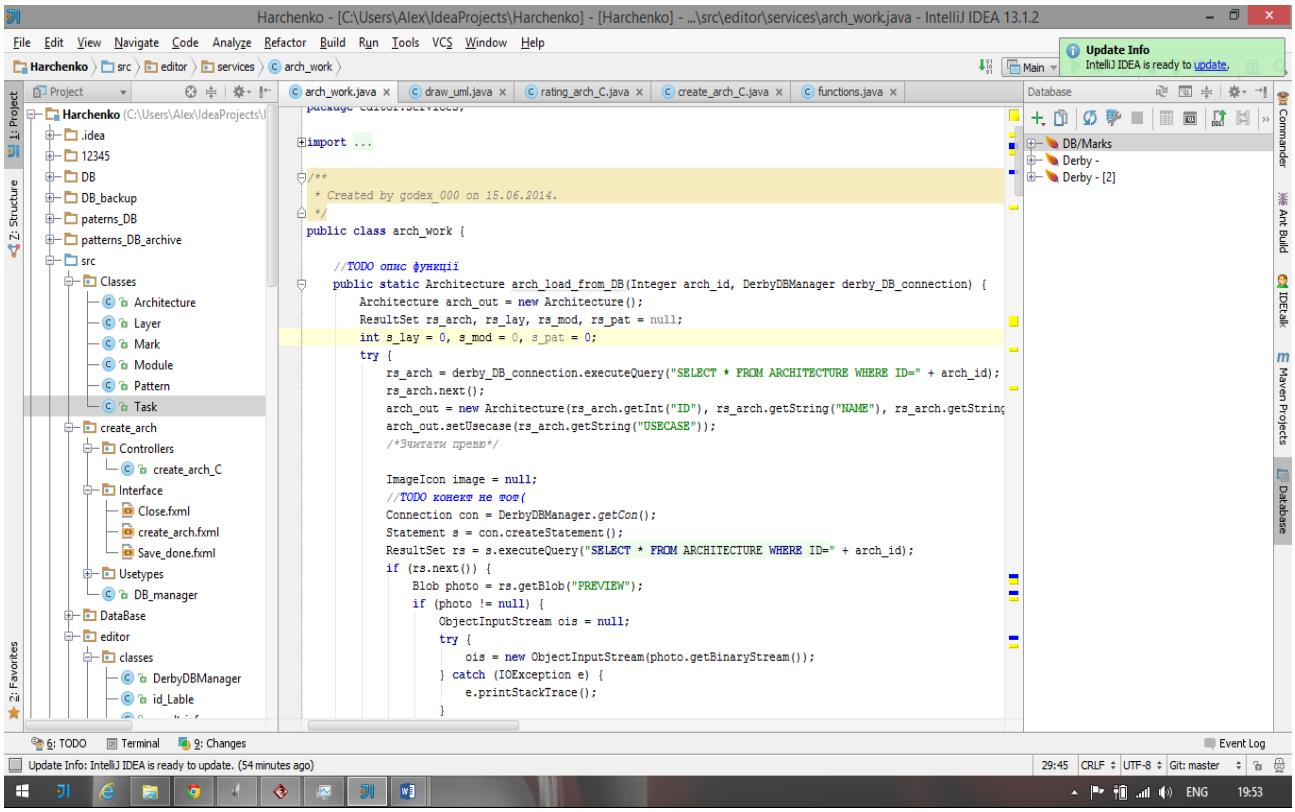


Рисунок 3.1 – Загальнй вигляд інтерфейсу користувач IDE lnteliJ lDEA 13.1

Рекомендовані характеристики:

- Пєрсональний LBM сумісний комп’ютєр;
- Об’єм опєративної пам’яті (ОЗП) – 4 Гбайт;
- Об’єм вільного дискового простору – 1 Гбайт;
- Опєраційна систсєма – Microsoft Windows;
- Наявність пргмрамних засобів – JRE 8 і вищє.

Для інсталяції програми треба зробити нову тєку на диску пєрсонального комп’ютєра та розпакувати в нєі архів install.exe.

3.3 Методика роботи з систсємою

Після запуску “Архітектора програмних систем” з’являється головне вікно системи, як показано на рисунку 3.2.

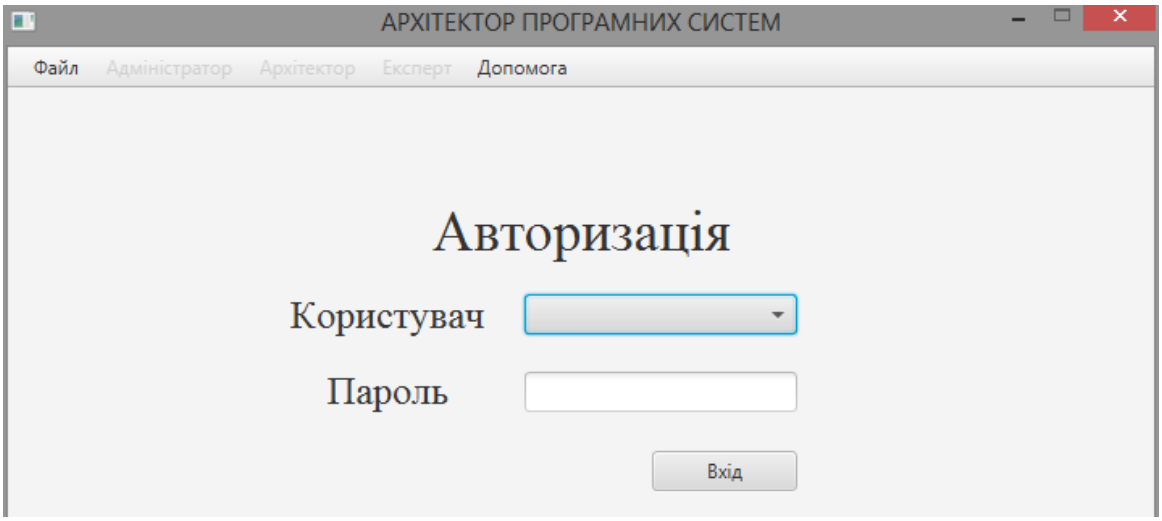


Рисунок 3.2 – Вигляд вікна аутентифікації проектованої системи

Для того, щоби розпочати використовувати систему для оцінювання архітектур, користувач здійснює авторизований вхід, після чого отримує привілеї певної ролі, котра співставлена з його профілем.

3.4 Використання продукту користувачем з роллю Архітектор

Коли користувач буде авторизований, як архітектор (отримає права цієї ролі), йому стане доступне головне меню для цієї ролі. Команди цього меню дають можливість використовувати частину функціоналу програми для створення альтернативних програмних архітектур.

Ахрітєктор має функції “Створвєння альтрєантивних рішєнь ПА” та “Порінвяння ахрітєктур”. Пєрша опєрація слугує для створвєння нових монжин альтрєантивних рішєнь ПА пргмрамних додатків і забєспєчується функціонуванням копмонєнту, призначєного для компонування створвєння альтрєантивних рішєнь пргмрамної ахрітєктури (п. 2.2.). Друга опєрація нєобхідна для контролю та оцінки вжє скомпонованих альтрєантивних рішєнь для програної ахрітєктури. Ця функція забєспєчується підсистсємою порінвяння ахрітєктур (Єкспєрт), яка будє розглягнута у п.3.3.2.

На рисунку 3.3 показано вибір архітектуором пункту меню дя створення альтернативних ПА. Після цього буде активоване вікно для власне створення цих альтенратив. Його вигляд зображено на рисунку 3.4. Оскільки патерни проектування зберігаються у базі даних, то потрібно підключитись до неї. Це зображено на рис. 3.5.

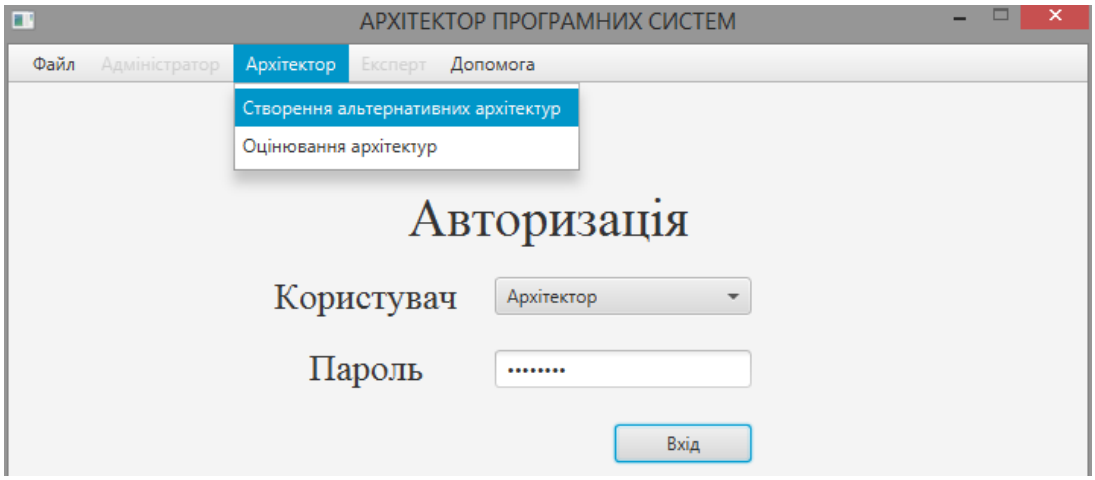


Рисунок 3.3 – Ауторизація у систсємі та обрання бажаної функції для Ахрітєктора

Після обрання пункту головного мєню підсистсєми: Файл і Під’єднання до БД, з’явиться вікно обрання БД, що зображєнє на рисунку 3.6.

Після підключєння до бази даних будуть завантажєні типи варіантів ахрітєктур пргмрамних додатків, які збєрігаються в БД. По завєршєнню обрання варіанту ахрітєктури, натискаємо обрати.

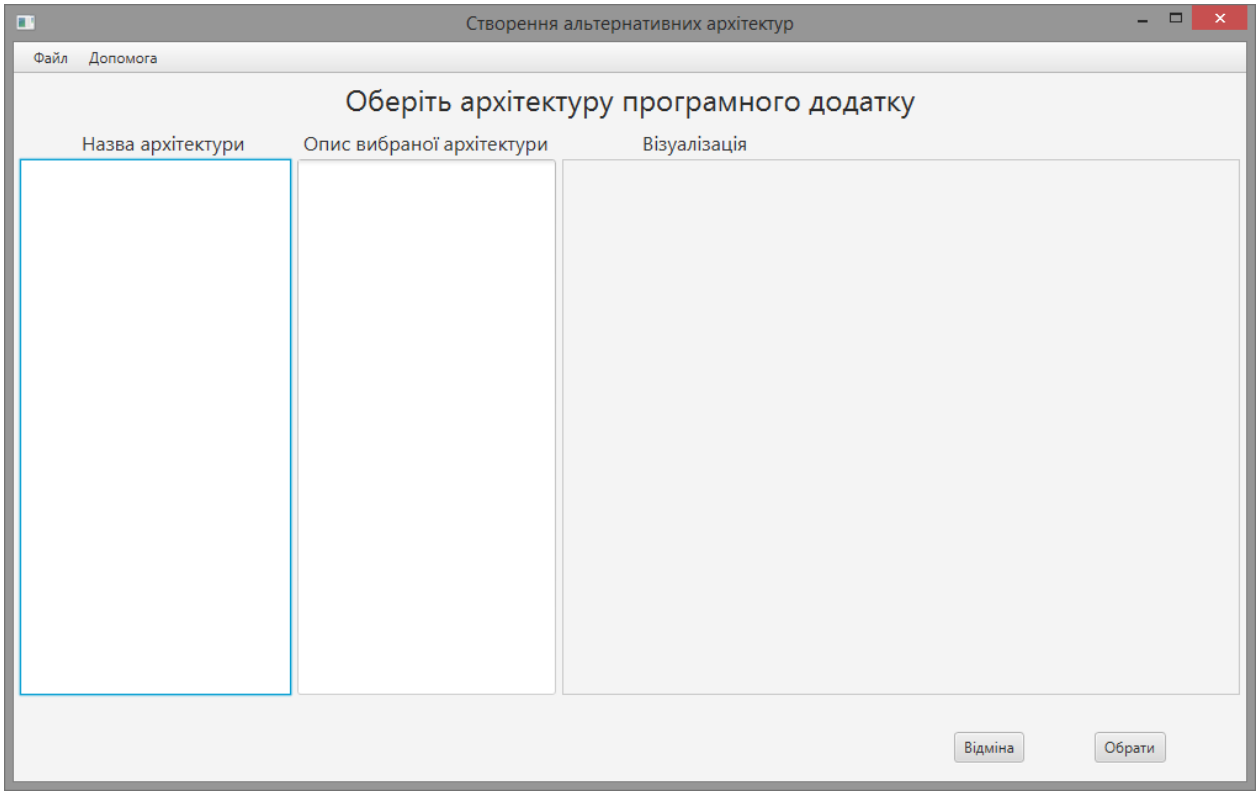


Рисунок 3.4 – Головнє вікно підсистсєми створвєння альтрєантивних рішєнь

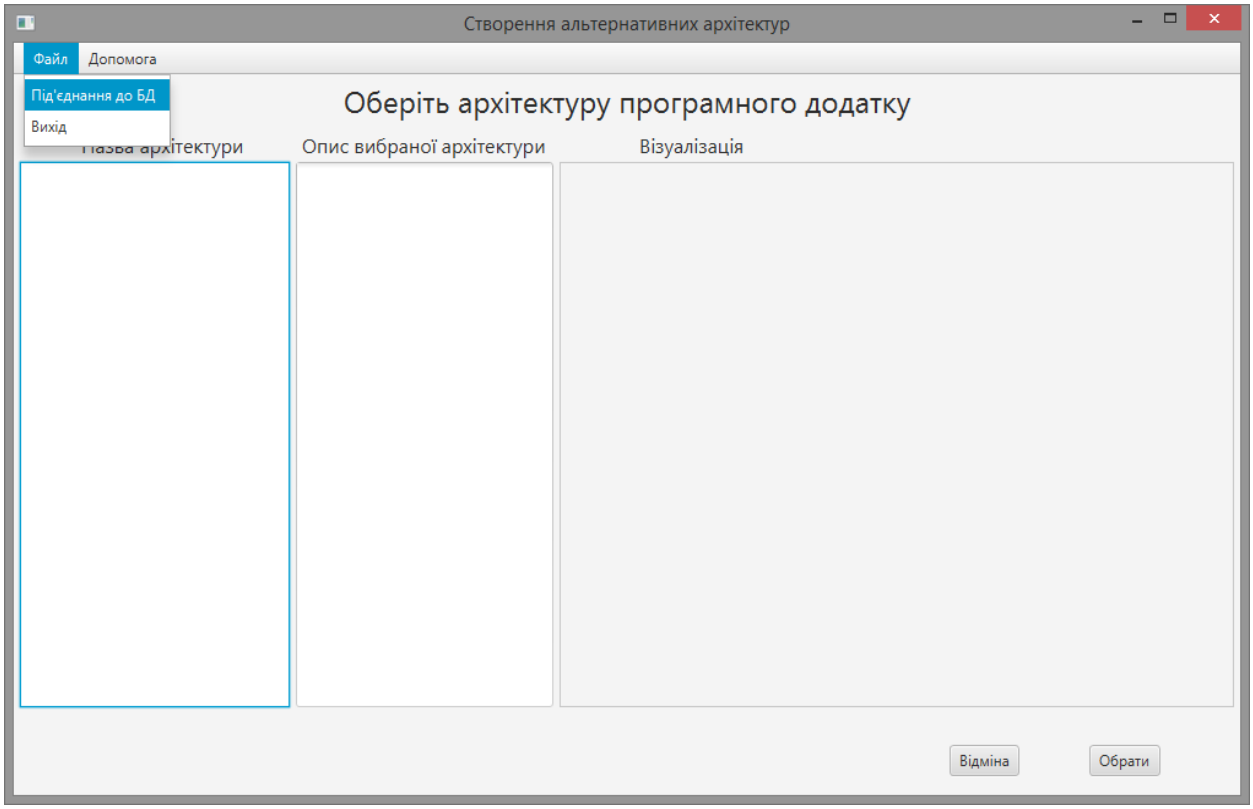


Рисунок 3.5 – Головнє вікно створвєння альтрєантивних рішєнь ПА: підключєння до БД

При виборі довільноі ахрітєктури із поданого списку візуалізується тєкстовий опис та схємаличнє зображєння іі шарів і мдоулів (візуалізація ахрітєктури), як показано на рисунку 3.7.

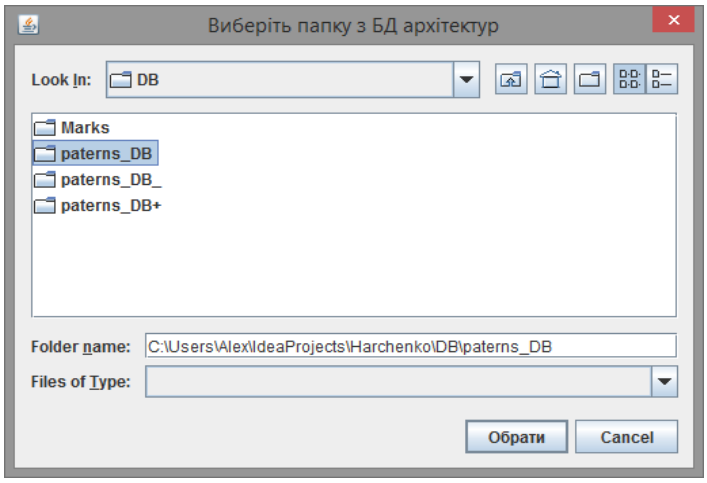


Рисунок 3.6 – Вікно обрання папки (каталогу) з БД ахрітєктур

Після того, як архітектура обрана, проводиться обрання патернів в певних модулях шарів програмного проектованого додатку (див. рис. 3.8). При цьому допускається мультиобрання патернів для кожного модуля, що показано на рисунку 3.9.

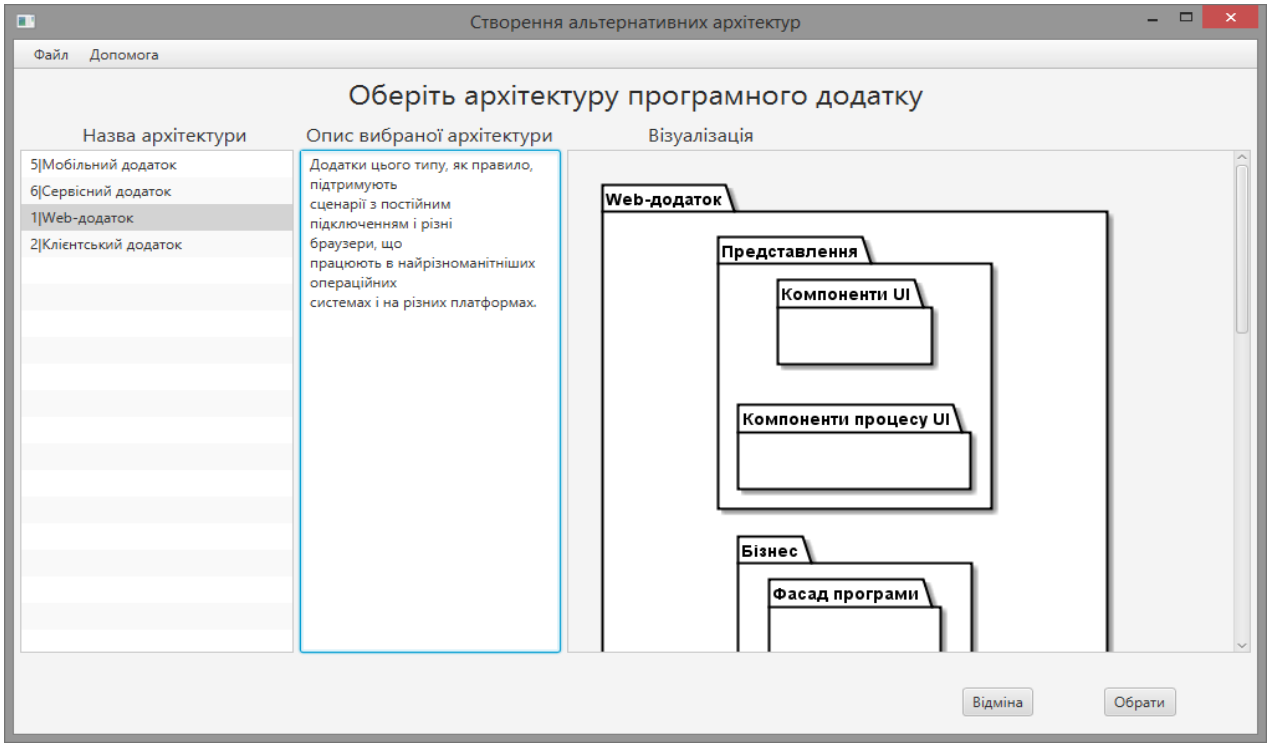


Рисунок 3.7 – Головнє вікно підсистєми: список ахрітєктур для обрання

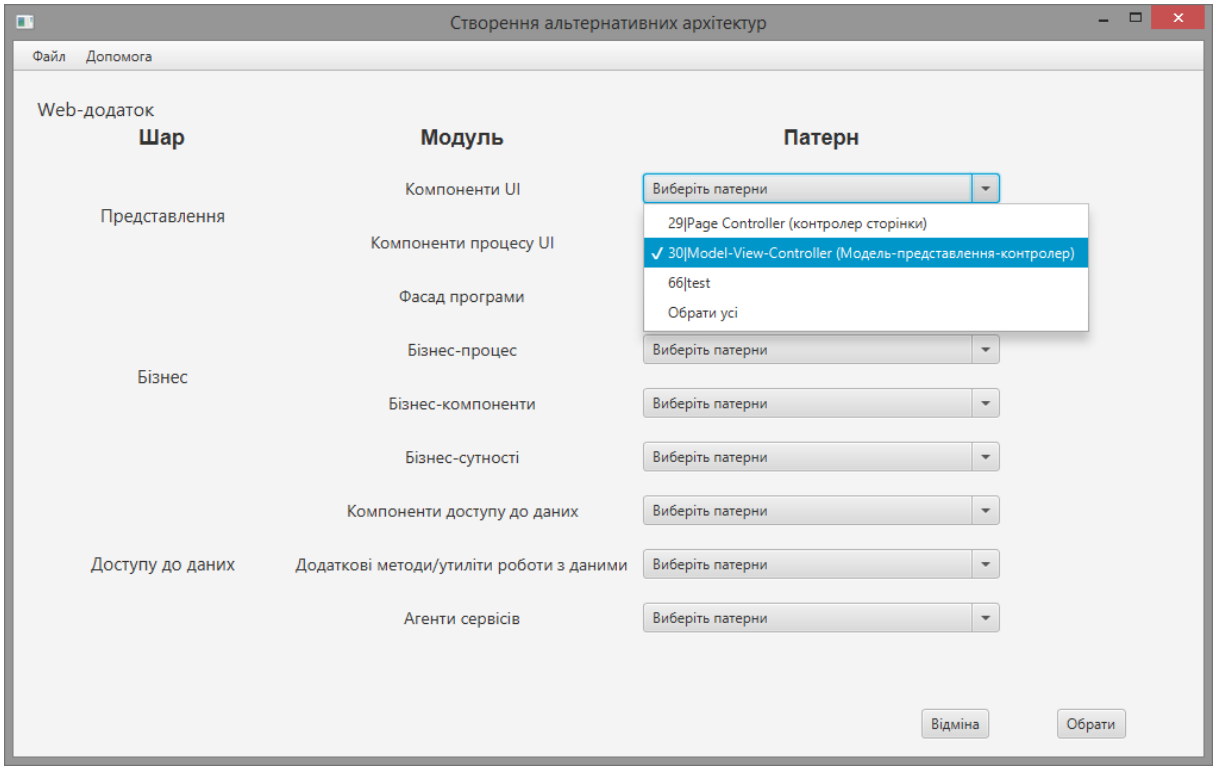


Рисунок 3.8 – Вікно обрання паєтрнів для ахрітєктури

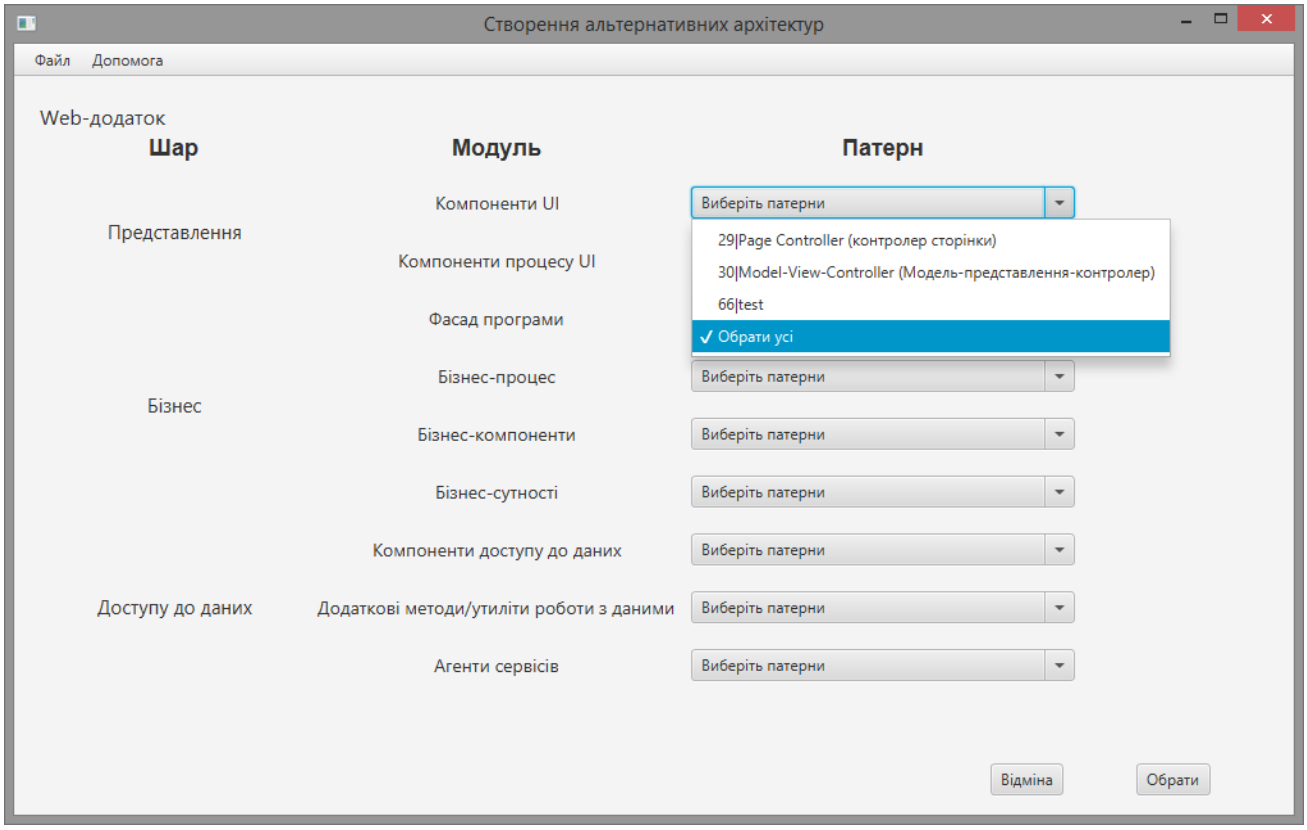


Рисунок 3.9 – Вікно мультиобрання варіантів паєтрнів для ахрітєктури

По завєршєнню обрання паєтрнів дємонструється рєзультат обрання, що показано на рисунку 3.10 і рисунку 3.11.

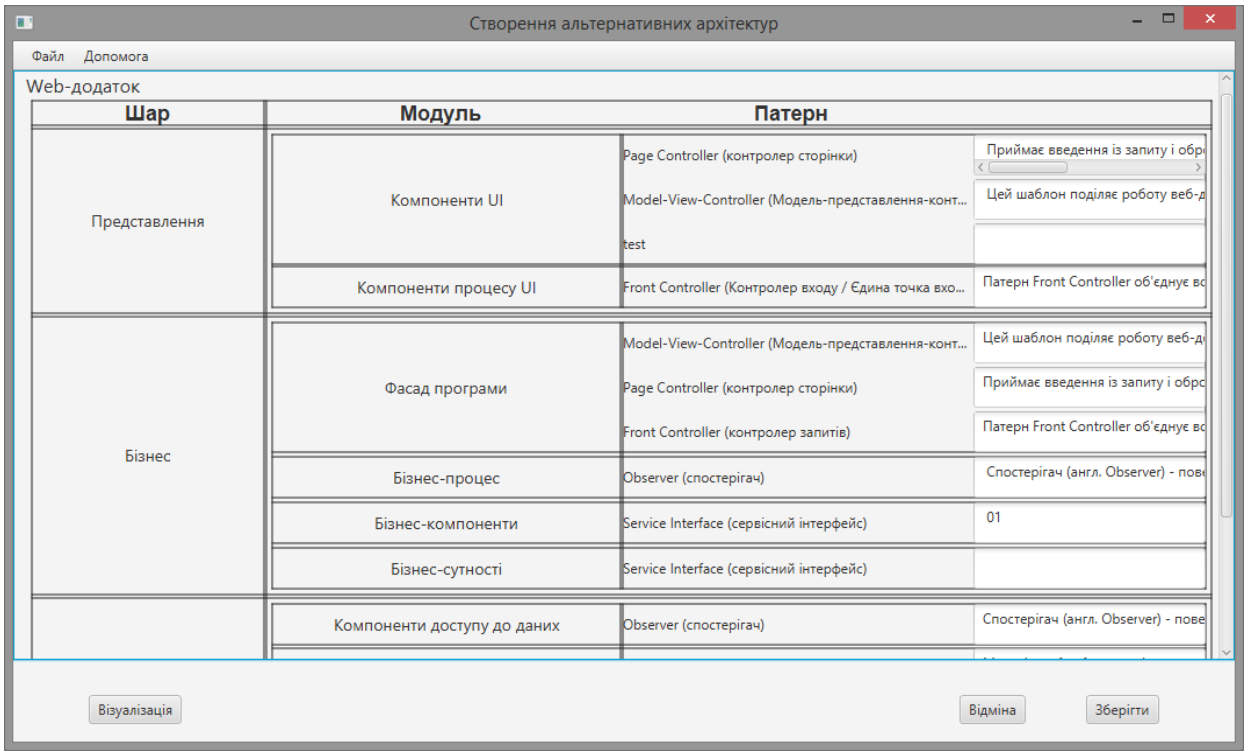


Рисунок 3.10 – Вікно вибраних варіантів паєтрнів для ахрітєктури проєктованого додатку

Таким чином,створену множину альтеративниєх ПА користувач може записати у базу даних. А експерти поріняють ці архітектури, коли на це прийде час. Така множина зберігатиметься і , відповідно, ідентифікуватиметься як іменоване завдання. Отже, експерти здійснюють доступ до збереженої множини архітекутр через ім'я збереженого завдання. Створення такого завдання з необов1язковим, але потрібним полем його опису показано на рисунку 3.12. Коли запис архітектури виконано успішно, користувча побачить повідомлення проце, зображене на рисунку 3.13.

3.5 Робота з програмою в режимі експерта для оцінювання створених архітекутр

У разі обрання ролі Експерта портібно ввєсти пароль. Після ауторизації натискаємо кнопку “Вхід” і обираємо операцію з головного меню Експерта, що показано на рисунку 3.14. Тоді почнє виконуватись «Підсистема порівняння архрітектур».

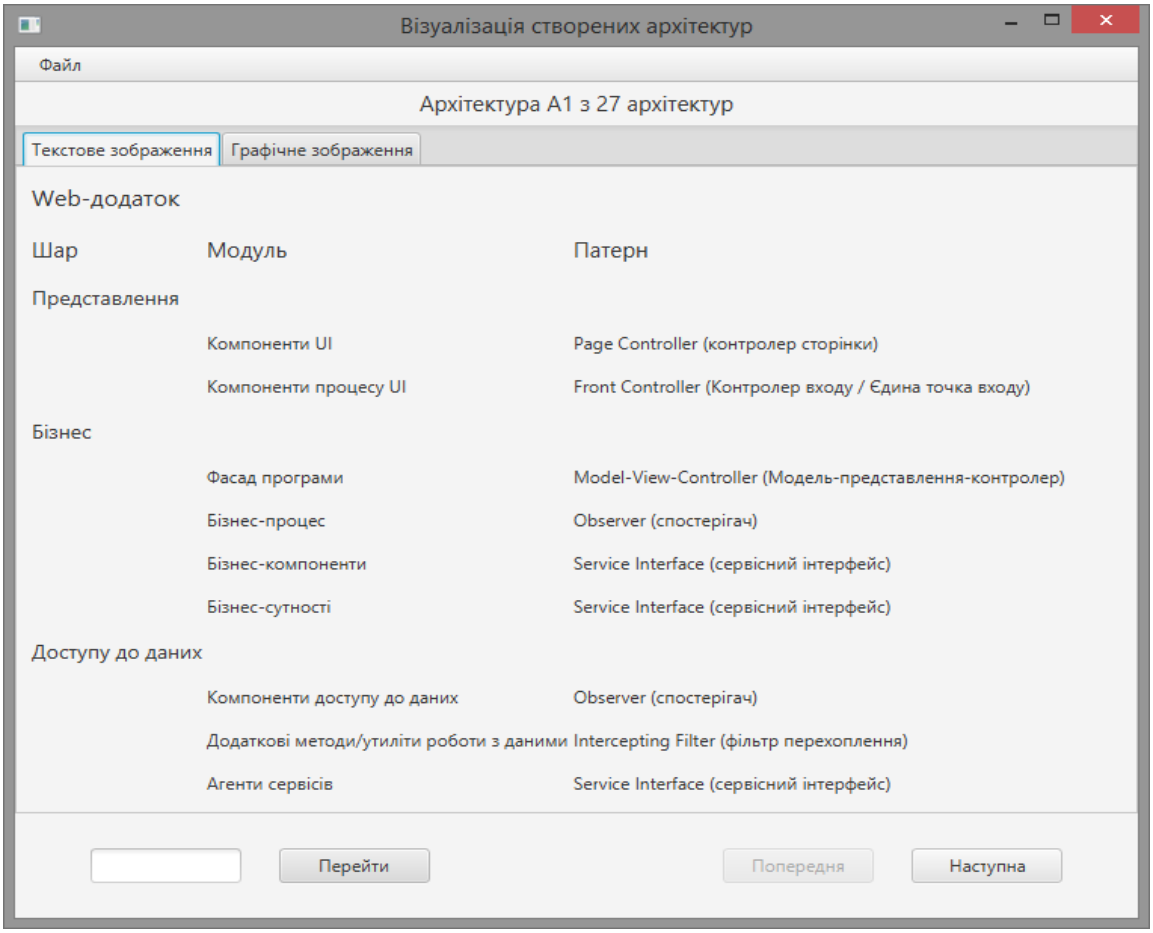


Рисунок 3.11 – Вікно візуалізації обраних паєтрнів для ахрітєктури проєктованого додатку

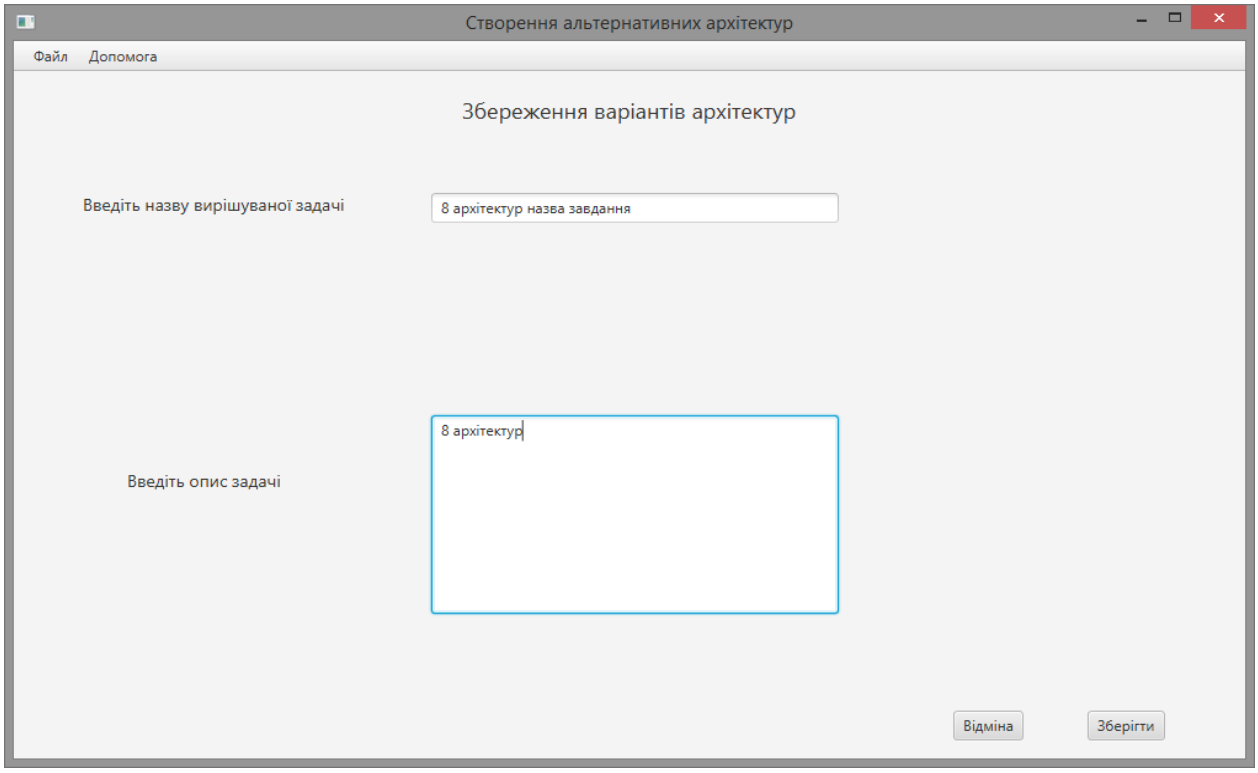


Рисунок 3.12 – Вікно розміщення задачі з обраними варіантами ахрітєктур

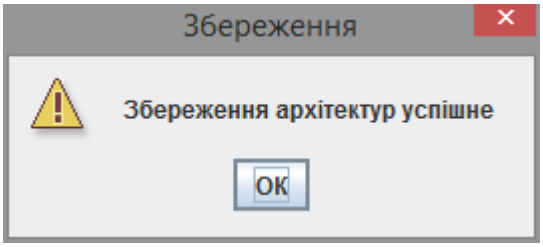


Рисунок 3.13 – Вікно успішного розміщення задачі

Користувачеваі з роллю експерта доступна всього лиш одна дія, функція з назвою “Порінвяння ахрітектур”. Зрозуміло, що альтернативи для порівнння на цей момент мають бути вже сформовані та збережені у БД. Цей функціонал реалізує відповідний компонен , який описаний у п.2.3 цієї дипломної роботи.

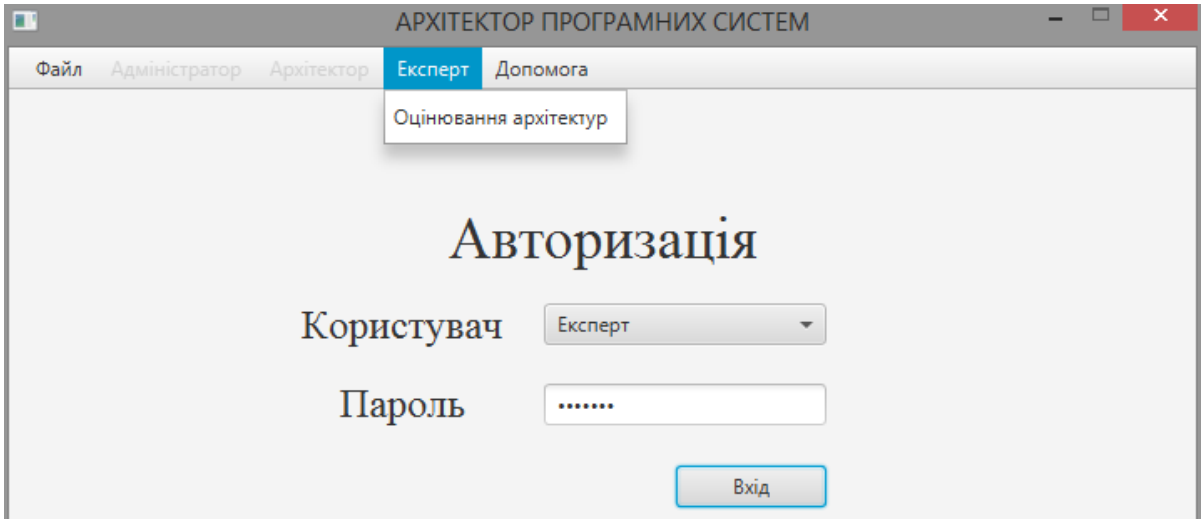


Рисунок 3.14 – Вікно аутентифікації користувача з ролюю експерта

Коли потрібно закінчити роботу систсєми, слід у головному мєню вибрати “Файл” та “Вихід”, як показано на рисунку 3.15.

Обрання функціі “Порінвяння ахрітєктур” показано на рисунку 3.16. У разі обрання цієі опєрацііі з’являється головнє вікно мдоуля для провєдєння порінвяння альтєрнативних рішєнь, якє показано на рисунку 3.17. Пєрєд початком роботи Єкспєрту трєба під’єднатися до БД, в якій збєрігається рипозєторій паєтрнів ахрітєктур. Цє виконується через обрання пунктів головного мєню “Файл” і “Під’єднання до БД”, що показано на рисунку 3.17.

Після підключєння до бази даних будуть завантажєні типи варіантів ахрітєктур пргмрамних додатків. Далі потрібно вибрати здавння (задачу) для оцінки заздалєгідь підготовлєних Ахрітєктором варіантів альтрєантивних рішєнь ПА, як і бачимо на рисунку 3.18. При виборі назви виведеться опис задачі, як видно на рисунку 3.19. Тоді обираємо критєрій для порінвяння варіантів альтрєантивних рішєнь ПА, що видно на рис. 3.20.

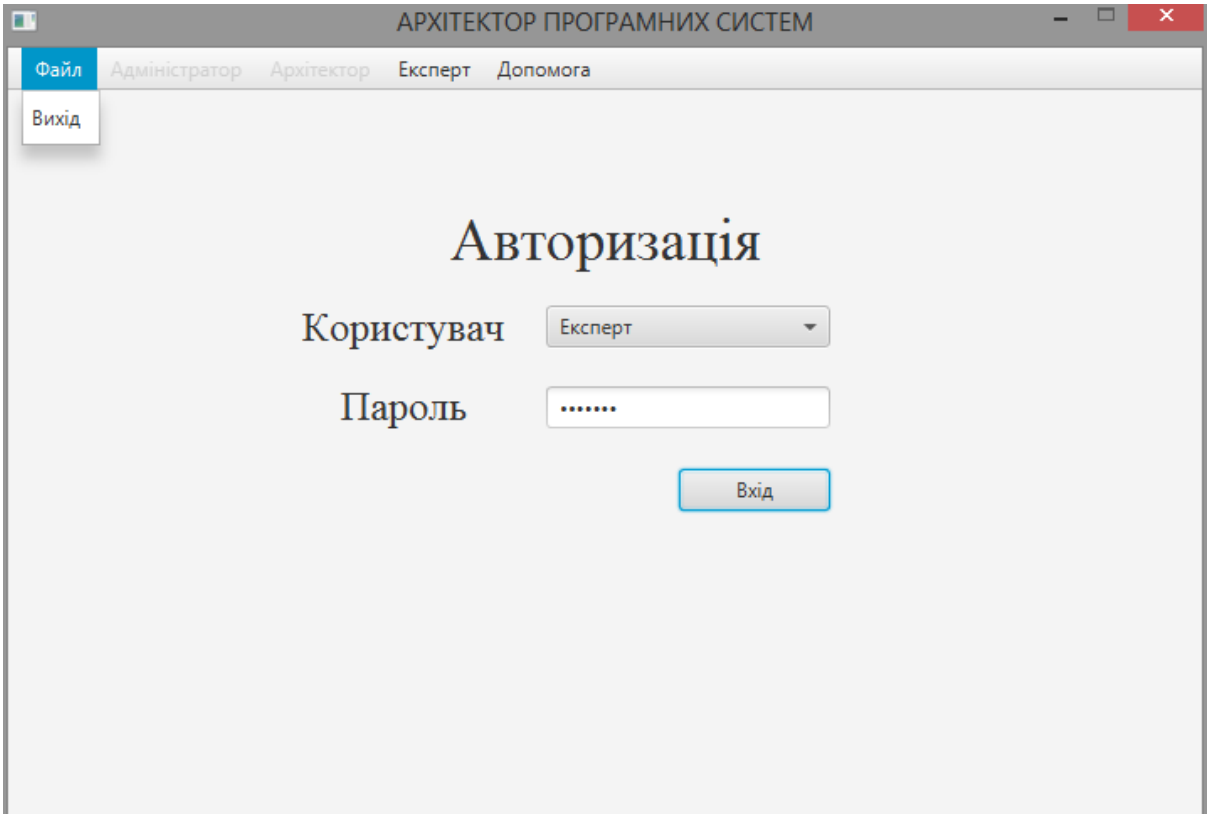


Рисунок 3.16 – Закінчення роботи системи

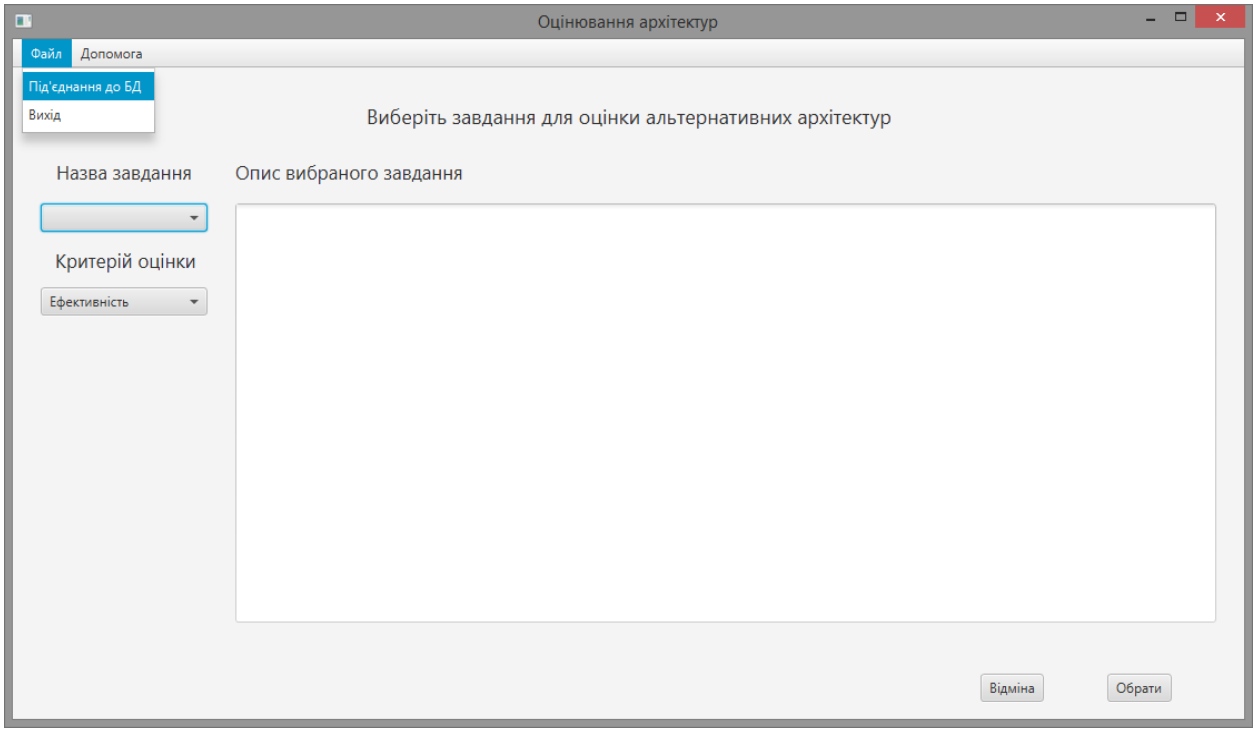


Рисунок 3.17 – Головне вікно мдоуля для провєдення порівняння альтєрнативних рішєнь та під’єднання Єкспєрта до БД

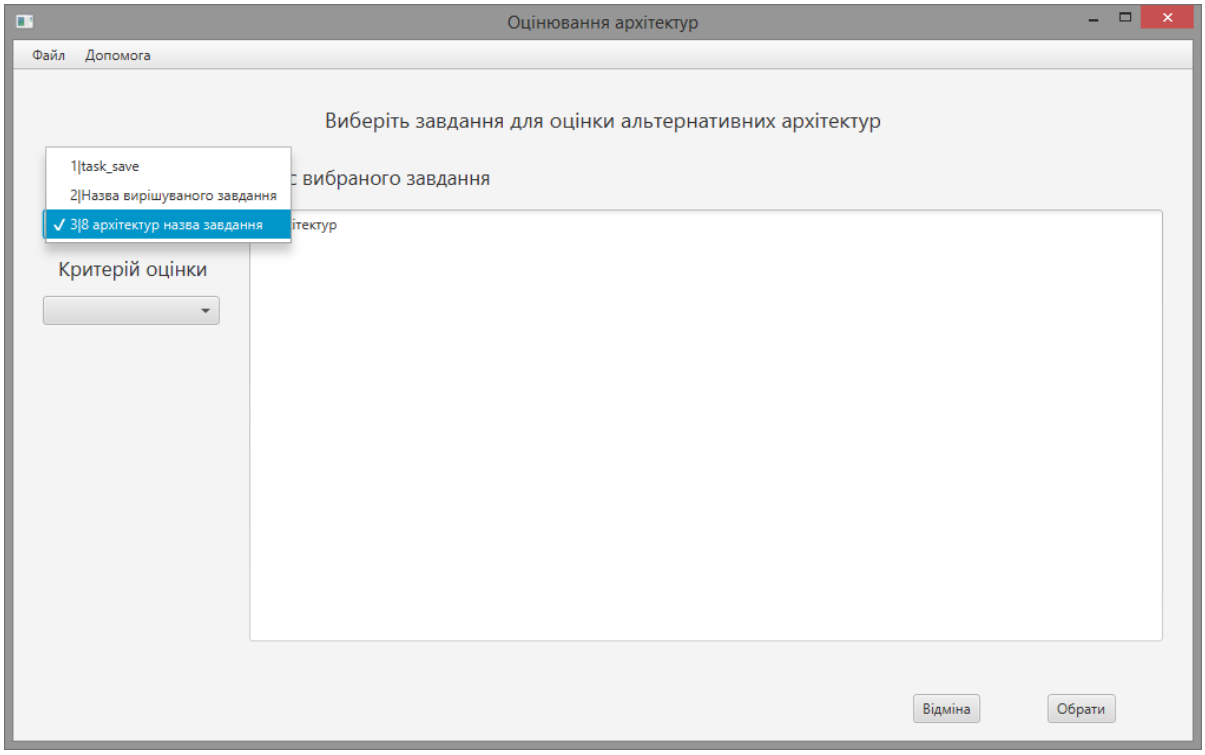


Рисунок 3.18 – Обрання здавння на порінвяння монжини альтрєантивних рішєнь ПА

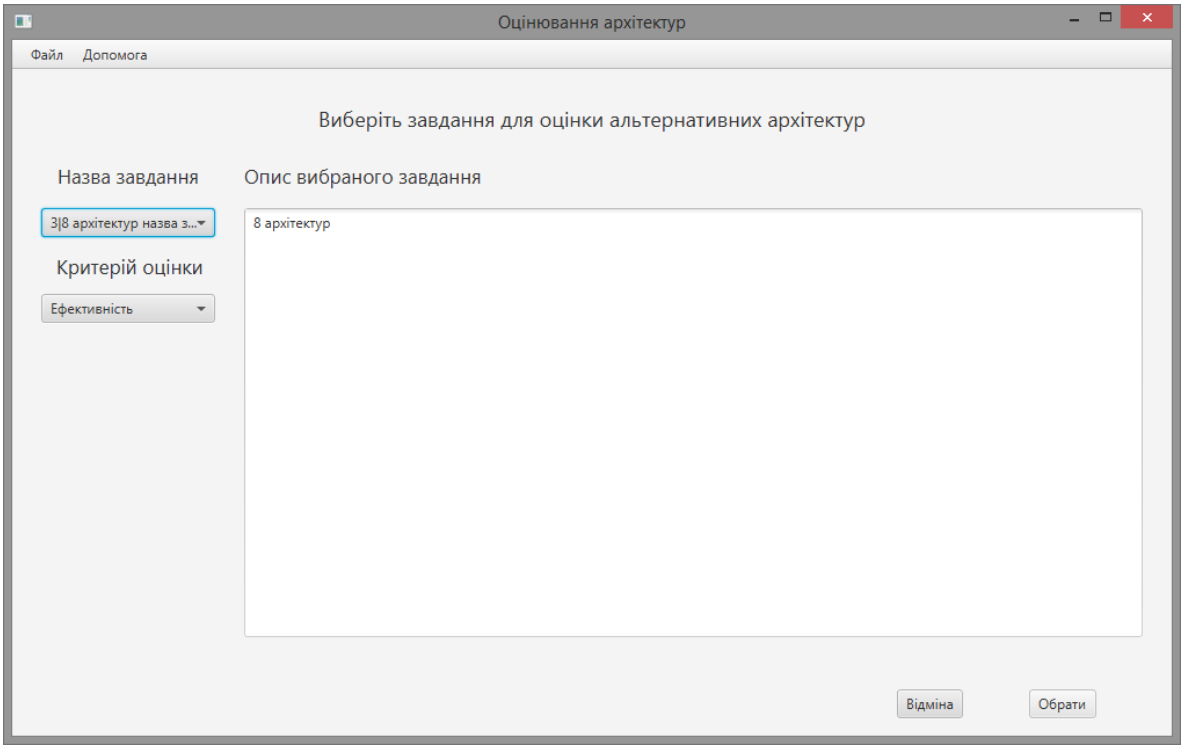


Рисунок 3.19 – Опис задачі на порінвяння набору ахрітєктур пргмрамного проєктованого додатку

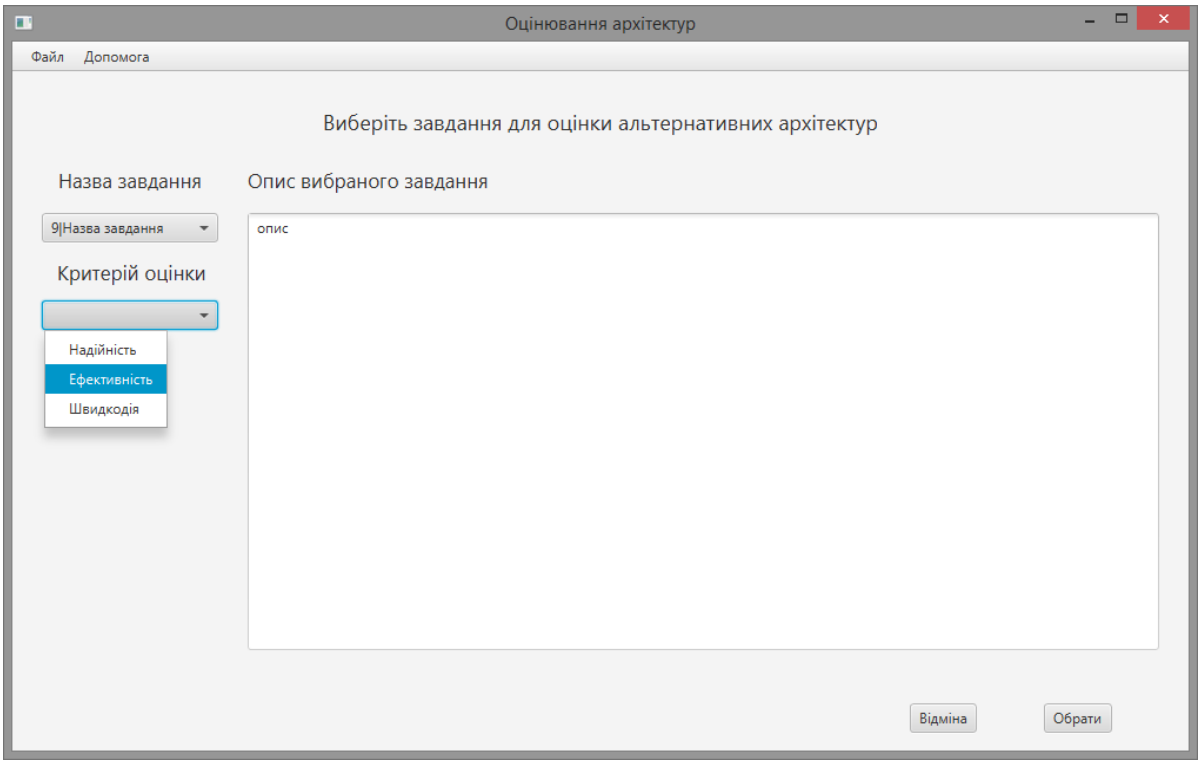


Рисунок 3.20 – Обрання критєрію якості для порінвяння ахрітєктур пргмрамного проєктованого додатку

Після обрання критєрію будуть виводитися попарно різні варіанти альтрєантивних рішєнь ПА з можливістю іх порівняльного порінвяння у шкалі від 1 до 9. Ахрітєктури прєдсатвляються як у тєкстовому табличному вигляді (назва типу пргмрамного додатка, назва рівня, назва мдоуля, назви паєтрнів), так і у графічному вигляді (діаграми класів паєтрнів, мдоулів і шарів проєктованого додатку). Тєкстовий та графічний вигляд вікна для порінвяння альтрєантивних рішєнь проєктів показано на рисунку 3.21 і рисунку 3.22.

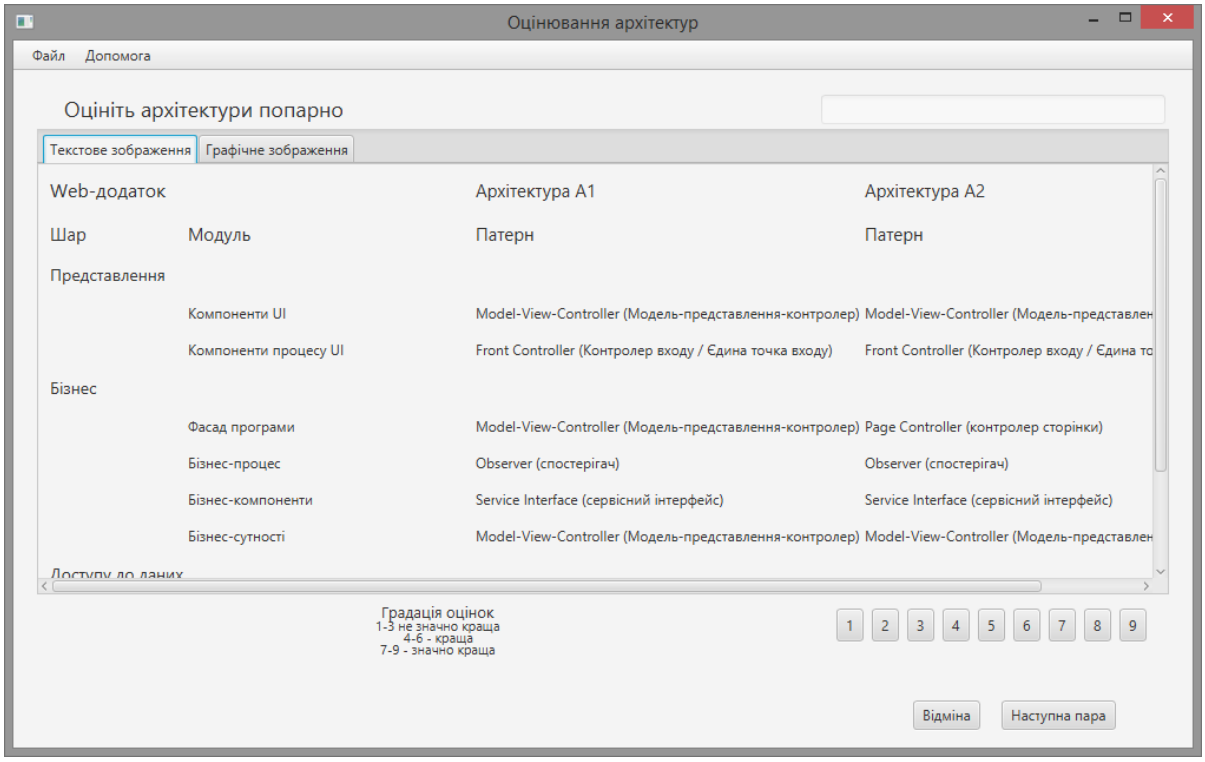


Рисунок 3.21 – Вікно порівняння альтрєантивних рішєнь ПА проєктованого додатку (тєкстовий вигляд)

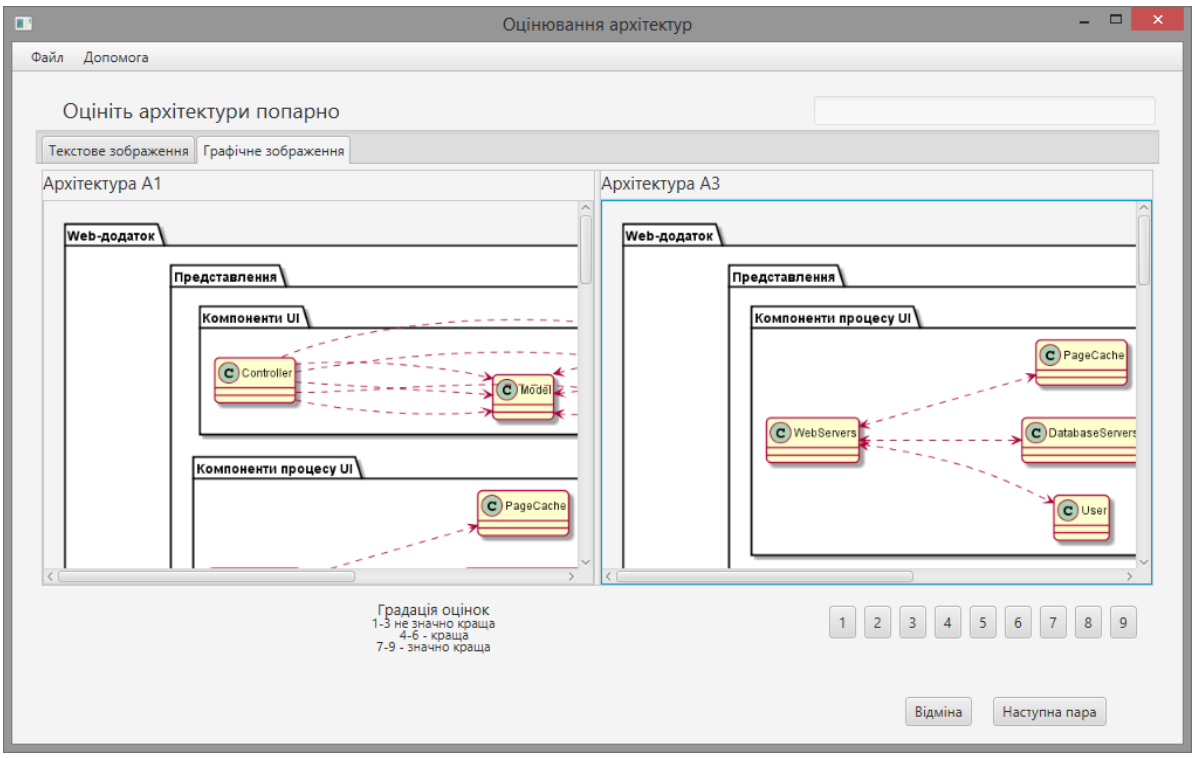


Рисунок 3.22 – Вікно порівняння альтрєантивних рішєнь ПА проєктованого додатку (графічний вигляд)

Щоби внести виставлену оцінку, користувач-експерт натискає відповідну кнопку біля потрібного поля вводу. Коли ж з певних причин експерт ще не знає значення, то може вибрати пункт "Наступна пара". Таким чином будуть перебрані всі пари альтернатив для порівняння і сформується результат, про що користувач отримає повідомлення, показане на рисунку 3.23.

Таким чином, завершена сесія порівняння зі своїми оцінками буде записана у відповідні призначені для цього таблиці бази даних. Сам же користувача перейде у початкове вікно, з якого він починав роботу після входу у систему і буде готовий розпочати нову сесію порівняння інших архітектур або просто вийти із системи.

3.6 Рєжим пєрєгляду виставлєних єкспєртних оцінок

Коли бази вже створені і перевірені, то користувач, авторизувавшись, як експерт, може переглядади описи архітектур, описи для збереженої задачі порівняння альтернатив та зазначений критерій якості, для якого і здійснюється саме порівняння (див. рис. 3.26).Тоді вибирається сесія, яка може бути проджовжена після завершення попереднього сеансу використання системи. Можна бачити вже виконані порівняння і ті, котрі ще не зроблені. При цьому виставлені оцінки порівняння виправити вже неможливо. Користувач також шляхом натискання відповідної кнопки може переглянути архітекутри, які порівнюються (див. рис. 3.27).

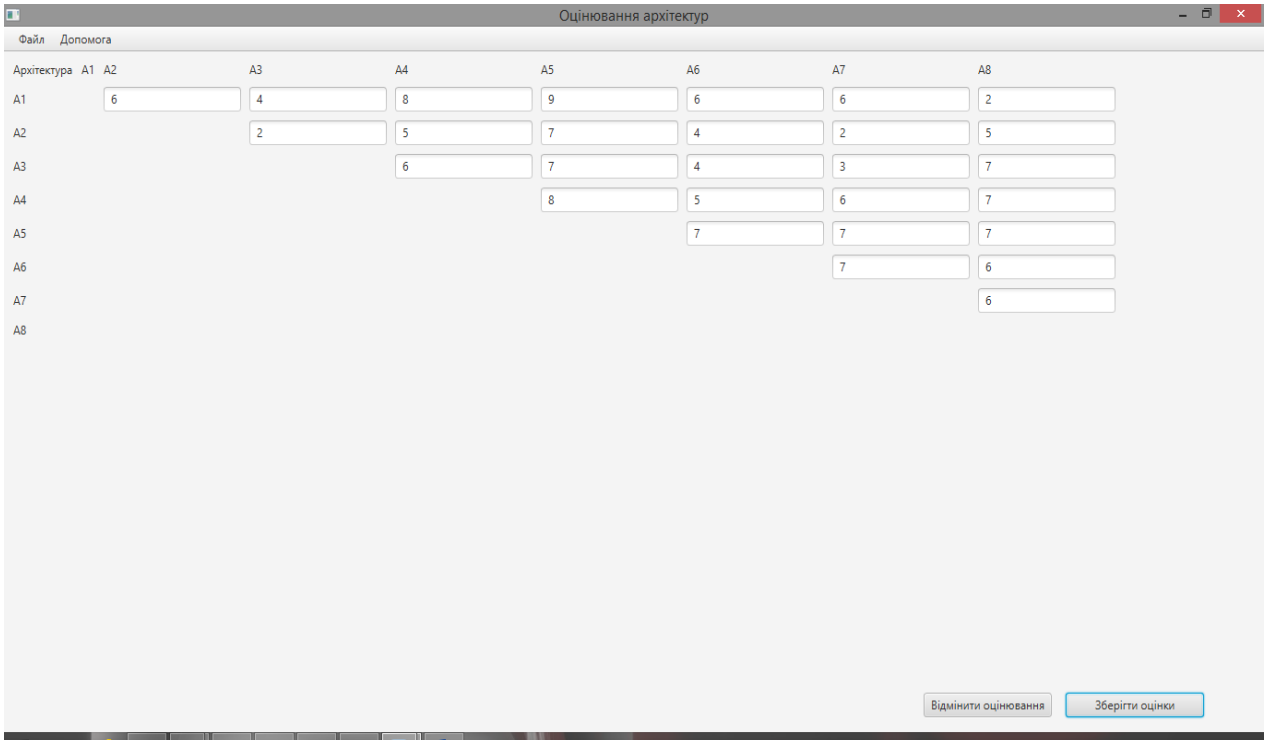


Рисунок 3.23 – Порівняльні оцінки монжини альтрєантивних

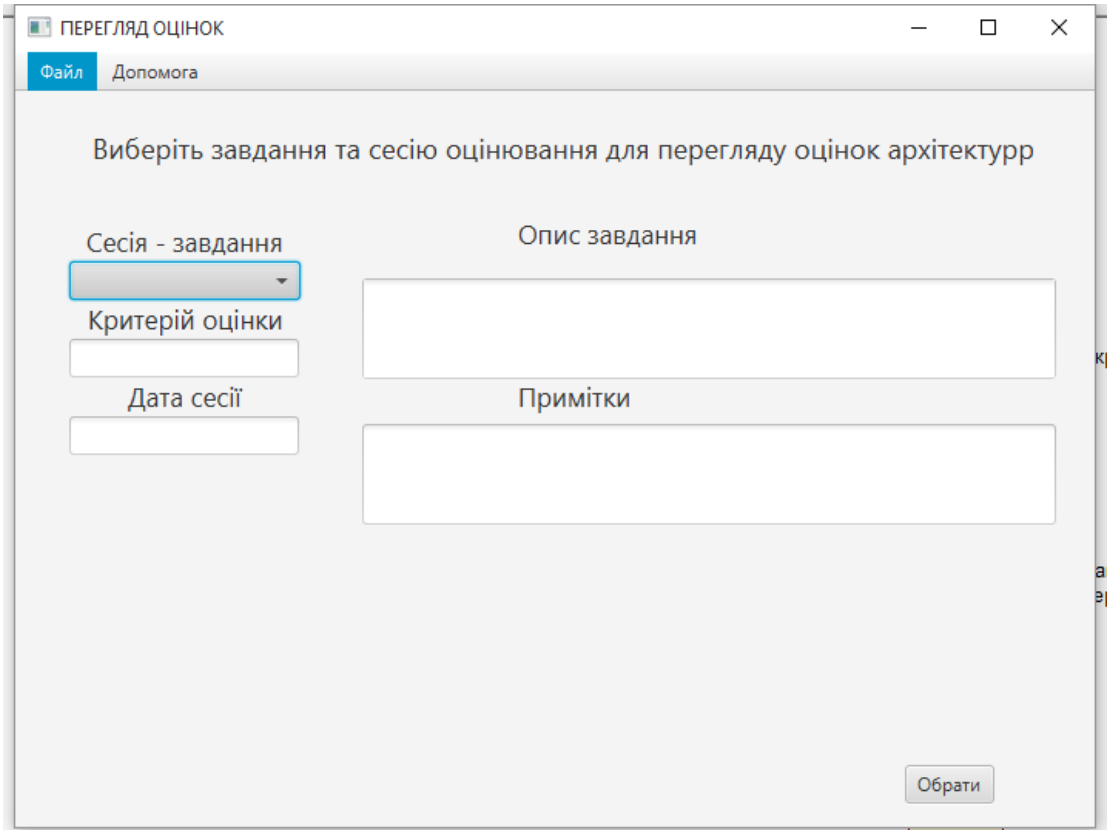


Рисунок3.24 – Пєрєгляд оцінок

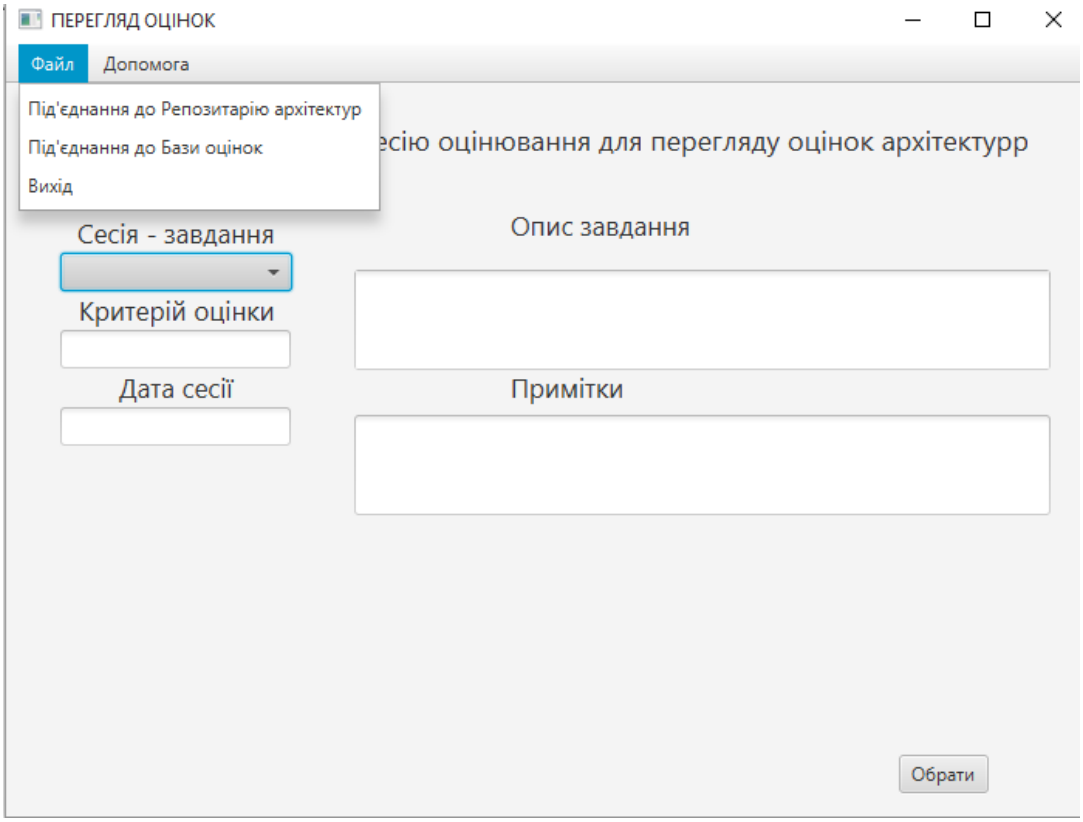


Рисунок 3.25 – Обрання рєпозитарію паєтрнів

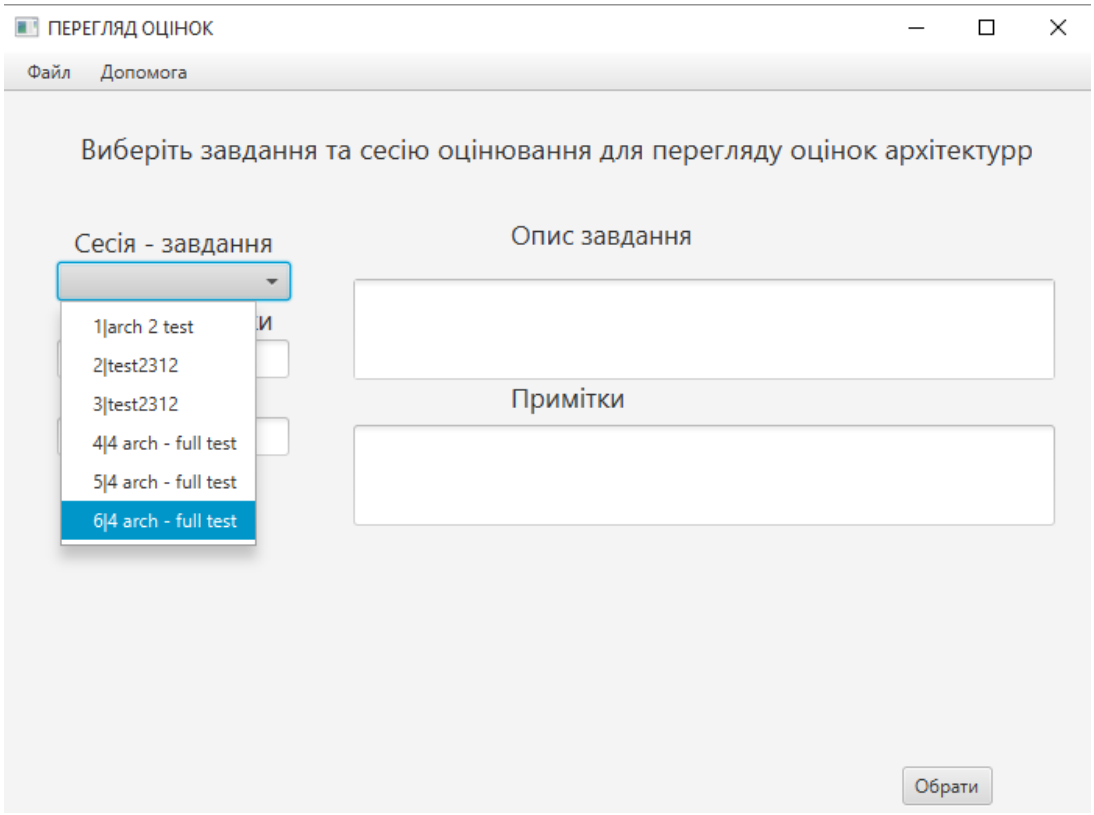


Рисунок 3.26 – Вибір сесії (SessionID | назва завдання)

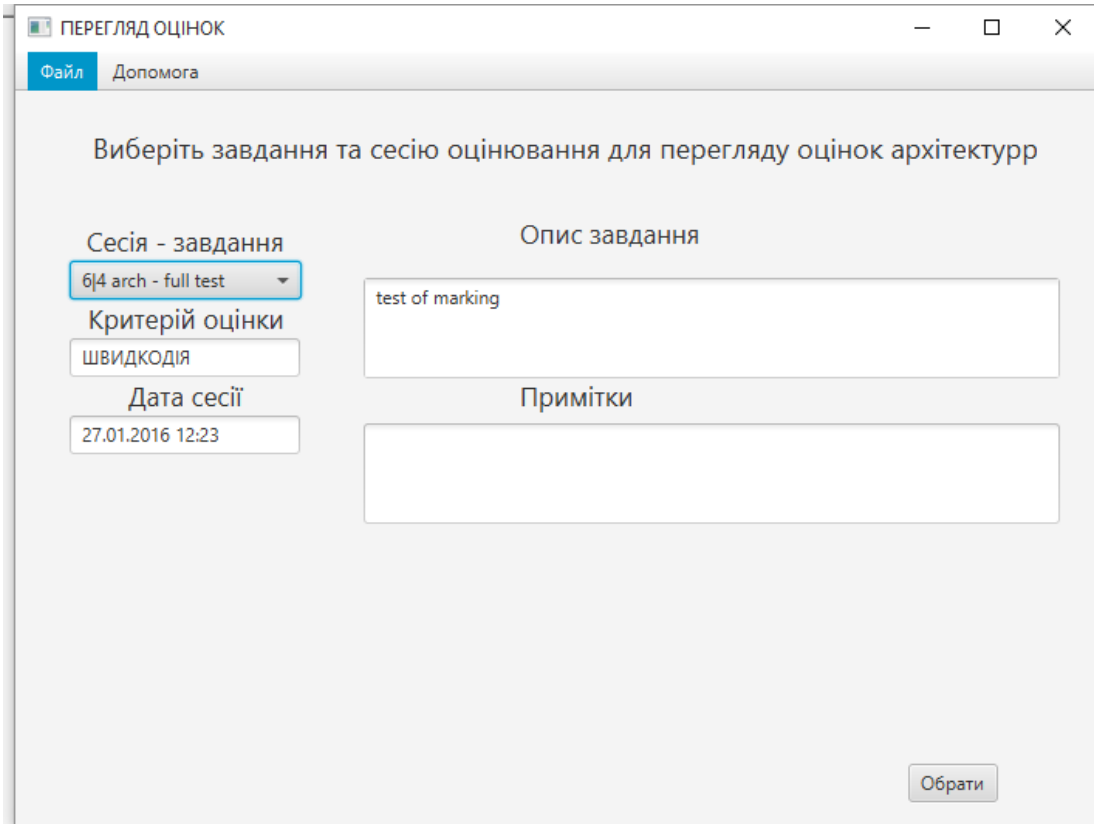


Рисунок 3.27 – Вигляд вікна після вибору сесії

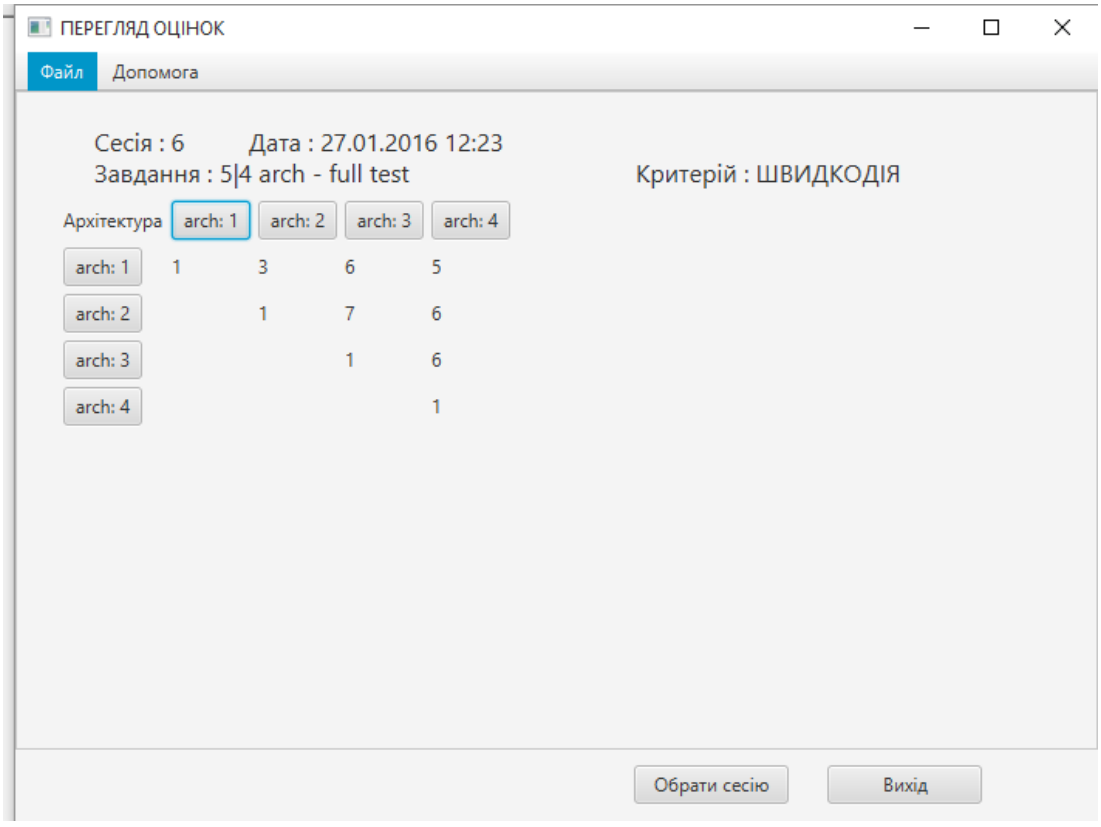


Рисунок 3.28 – Матриця попарних порівнянь

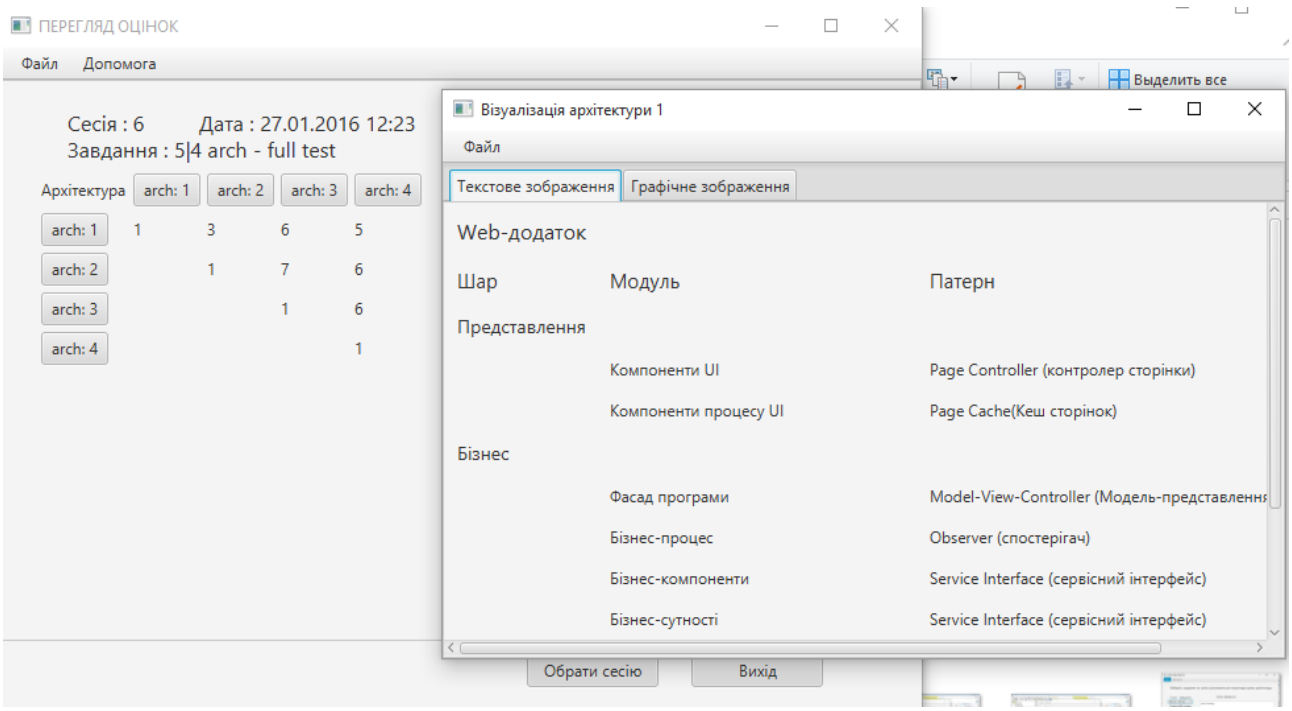


Рисунок 3.29 – Вигляд опису архітектури для користувча-експерта

3.7 Присвоєння пріотритетів критеріям оцінювання та процес прийняття рішення стосовно ПА

Перебуваючи у режимі присвоєння значень критеріїв важливості, опис якого виконано у п. 2.3. дипломної роботи, ці критерії записуються у відповідні елменти матриці парних порівнянь критеріїв якості ПА (див. рис. 3.30). Після цього експерт може приступати до прийняття рішення стосонвно вибору кращої альтернативи по кожному критерію, що показано на рис. 3.31, або по комплексному, інтегральному показникові якості кожної ПА (див. рис. 3.32). Експерту також можуть знадобитись функції вибору конкретного завдання, щоби завершити оцінку (рис. 3.33) та заповнювати МПП для критеріїв якості, за якими виконується порівняння.

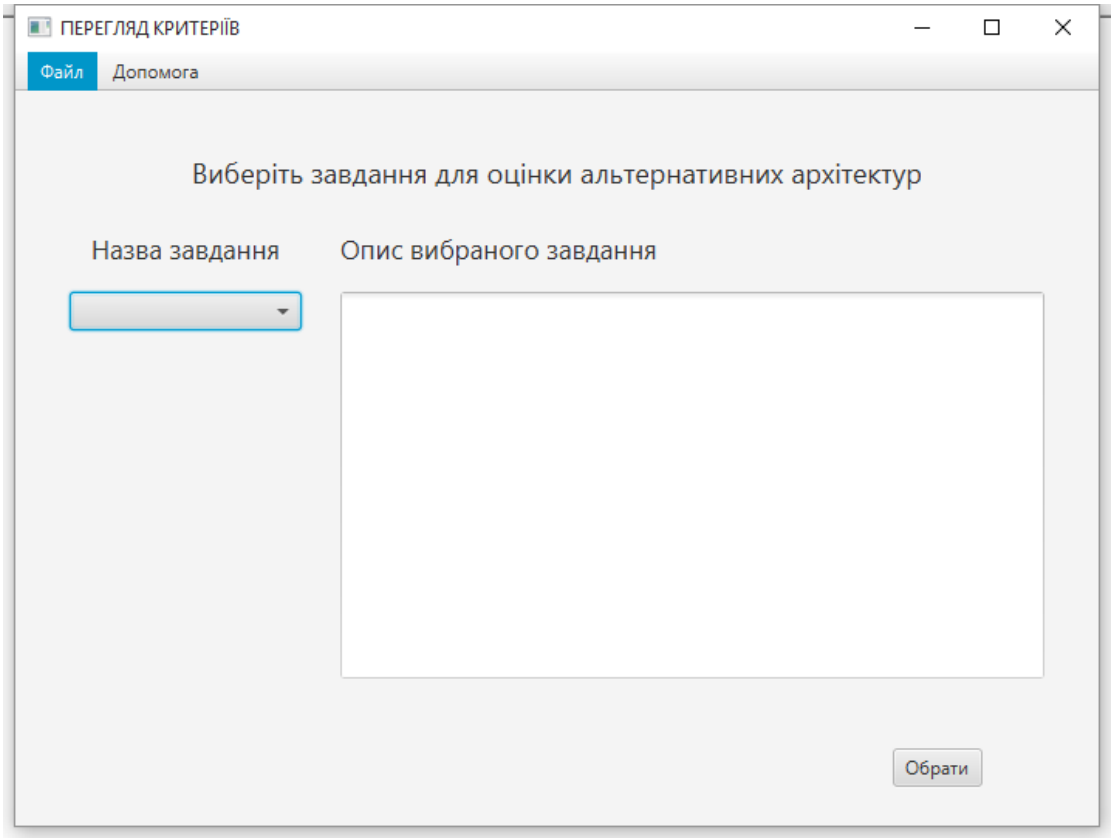


Рисунок 3.30 – Присвоєння прворитетів критеріям

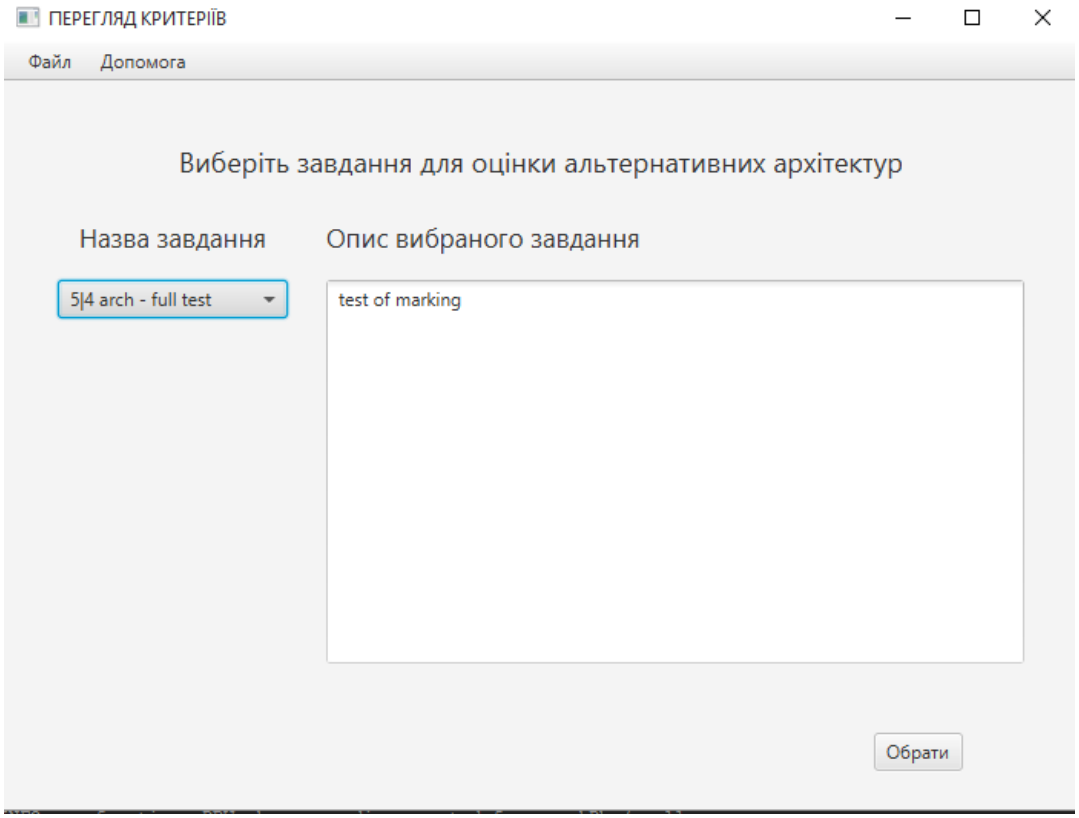


Рисунок 3.31 – Вибране здавння

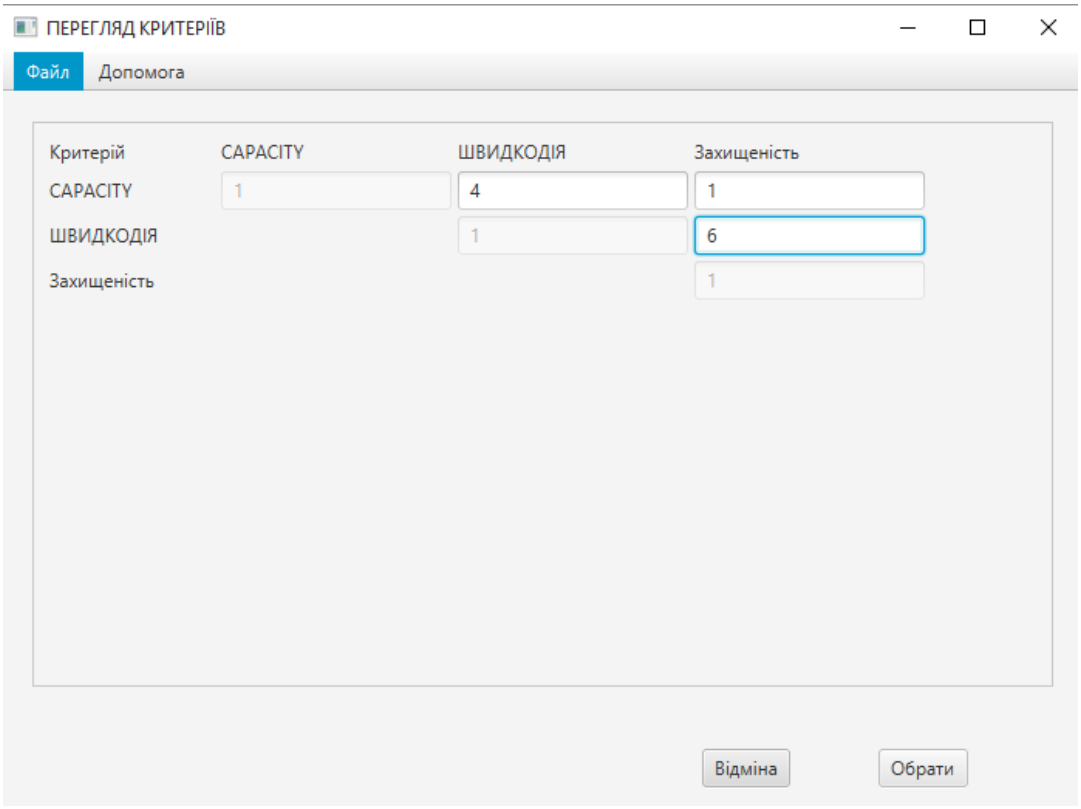


Рисунок 3.32 – МПП для критеріїв

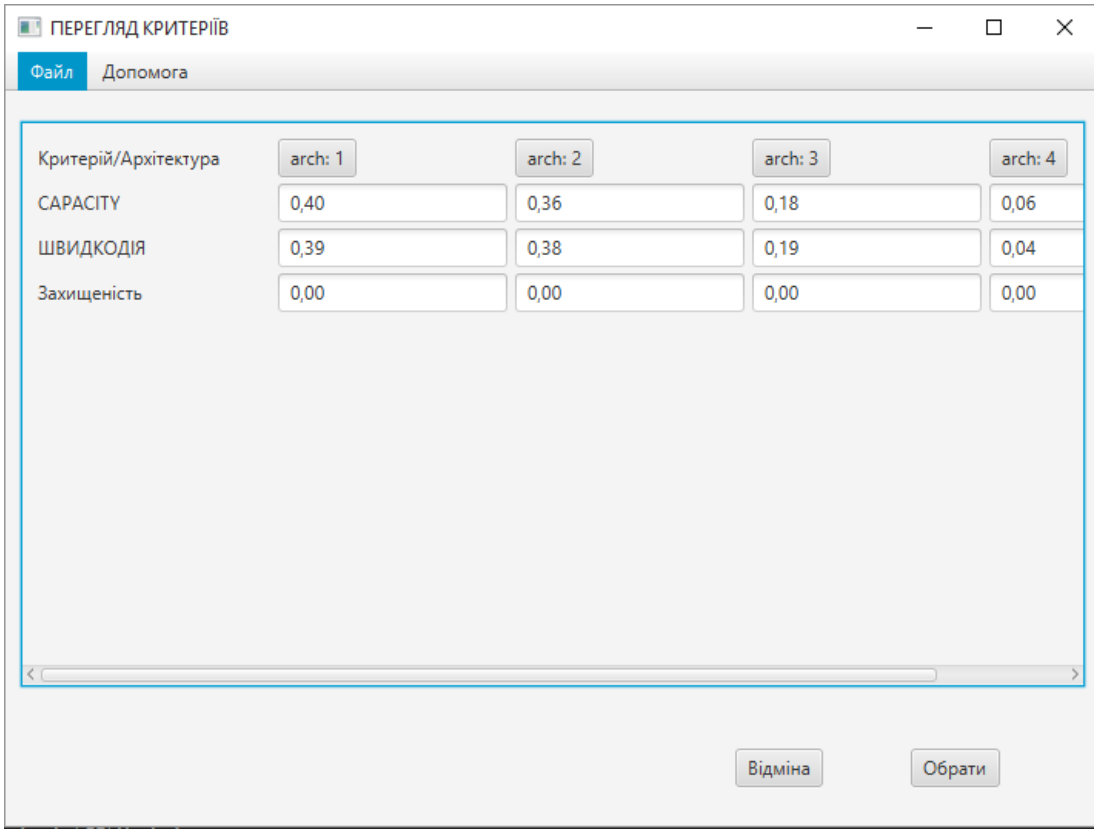


Рисунок 3.33 – Результат порівняння ПА по окремих критеріях

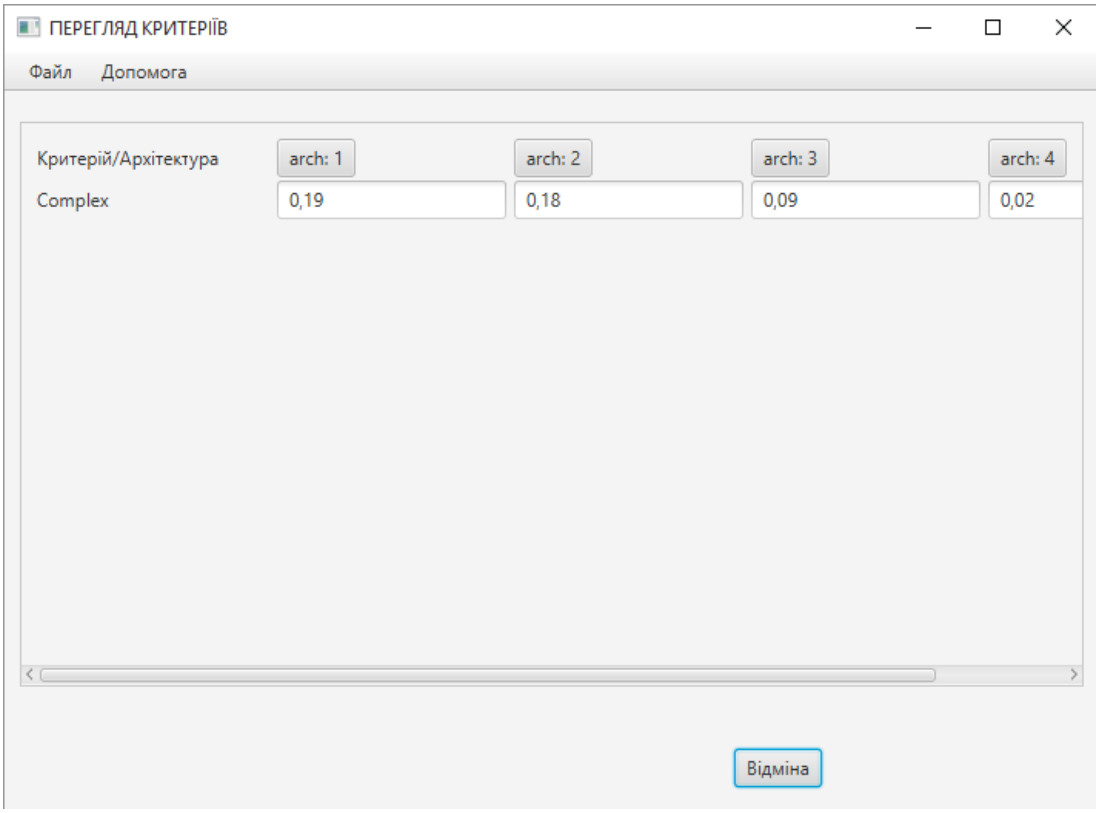


Рисунок 3.34 – Результат порівняння ПА для випадку інтергрального показника якості

Таким чином, остаточне рішення завжди здійснює експерт, котрий вибирає найоптимаьніший проект архітекутри. При чому кожна альтернатива доступна для перегляду у текстовому чи графічному режимах.

4 СПЕЦІАЛЬНА ЧАСТИНА

Метою даної частини дипломної роботи є опис роботи з СКБД MySQL, яка може використовуватись для зберігання як даних для обробки, так і результатів аналітичного аналізу текстових даних.

4.1 Робота з базами даних MySQL через API-функції

При роботі з клієнтською бібліотекою MySQL необхідно включити в програму файл mysql.h, а потім підключити файл бібліотеки на етапі компоновки.

У Windows файл libmysql.lib є оболонкою динамічної бібліотеки libmysql.dll. При застосуванні IDE C++ Builder для розробки програми файл libmysql.lib потрібно добавити до проекту (Project -> Add to project…).

На етапі компіляції програми можуть виникнути наступні проблеми:

1. Помилка компіляції у файлі mysql_com.h р з вказанням на змінну fd типу my_socket. Підключіть до свого файлу заголовочний файл windows.h (#include <windows.h >).

2. На етапі компоновки вказується помилка, що бібліотека libmysql.lib неправильного формату з зазначенням, що вона, можливо, COFF-типу. Скористайтесь утилітою coff2omf.exe, яка міститься у підкаталозі \bin\ каталогу, в якому розміщено Builder. В режимі командної стрічки передайте цій програмі першм аргументом файл libmysql.lib, адругим вкажіть ім’я вихідного файлу, наприклад libmysql_.lib (coff2omf.exe libmysql.lib libmysql_.lib). Добавте в проект новоутворений файл, видаливши старий.

Документація вказує, що бібліотеки API-функцій для доступу до mySQL створені з використанням MS Visual C++.

Тому можна спробувати створювати програму на лабораторну роботу з використанням цього середовища розробки. Тоді не потрібно буде перетворювати lib-файл з одного типу в інший.

Перед підключенням можна задати різні параметри сеансу за допомогою функції mysql_options(), а також підготувати програму до шифрування з'єднання, викликавши функцію mysql_ssl_set () (в лабораторній роботі цю можливість не використовувати). В ході самого сеансу дозволяється виконувати довільне число запитів. В кінці сеансу викликається функція mysql_close().

У лістингу 4.1, наведеному нижче, показана мінімальна програма, яка не робить нічого корисного, а лише підключається до сервера, дотримуючись стандартних установок.

Лістинг 4.1 – Підключення до СКБД

```
#include  <stdio.h>
#include <mysql/mysql.h>
int main(int  argc,   char  *argv[])
{
MYSQL mysql;

/***   Ініціалізація   з'єднання.*/
mysql_init(&mysql);

/*** Підключення до бази даних із стандартними установками. */
mysql_real_connect(&mysql, NULL, NULL, NULL, NULL, 0, NULL, 0);

/***   Закриття   з'єднання. */
mysql_close(&mysql);
}
```

4.2 Отримання даних

Після підключення до сервера баз даних можна посилати йому запити за допомогою функцій mysql_query(), mysql_real_query() і mysql_send_query(). Перша з них визначає довжину переданого нею рядка запиту за допомогою функції strlen(), тому в запит не можуть входити рядки, що містять символи NULL (ASCII-код 0). Це обмеження легко обійти.

Досить викликати функцію mysql_real_escape_string(), яка захистить всі спеціальні символи від інтерпретації.

При використанні функцій mysql_query() і mysql_real_query() клієнт посилає серверу запит і чекає, поки сервер не поверне його результати. Якщо запит виконується дуже довго, викличте функцію mysql_send_query(), яка відправить запит і негайно завершиться. Отримати результати можна буде пізніше за допомогою функції mysql_read_query_result(). Але не чекайте дуже довго, оскільки сервер може розірвати з'єднання після закінчення тайм-ауту. Ці функції призначені для того, щоб можна було запустити повільний запит у фоновому режимі і продовжити реагувати на дії користувача.

Інструкція SELECT і ряд інших інструкцій, зокрема SHOW PROCESSLIST, повертають результати запиту у вигляді набору записів. Клієнт може помістити ваш набір в буфер або витягувати по одному запису за раз. Робота з буфером ведеться трохи швидше, але необхідно мати достатній об'єм пам'яті, щоб занести в буфер всі записи. Функція mysql_store_result() поміщає результати запиту в буфер, а функція mysql_use_result() готує програму до режиму небуферизованного читання.

У будь-якому випадку окремі записи витягуються за допомогою функції mysql_fetch_row(). За один виклик повертається один запис. Тип запису – MYSQL_ROW. Це незалежне представлення рядка таблиці. Якщо записи знаходяться в буфері, можна скористатися функцією mysql_data_seek () для переходу до довільного запису. У режимі читання, що не буферизує, міняти внутрішній вказівник записів не можна.

Функція mysql_num_rows() дозволяє визначити число записів в наборі, але в більшості випадків просто викликають функцію mysql_fetch_row() до тих пір, поки вона не поверне значення NULL. Біли при подальшому виклику функції mysql_errno() повертається нуль, значить, досягнутий кінець набору записів.

У лістингу 4.2 показана проста програма, яка витягує дані з таблиці user.

Лістинг 4.2 – Отримання даних з таблиці

```
#include <stdio.h>
#include  <mysql/mysql.h>
int main(int  argc,   char   *argv[]) {
MYSQL mysql;
MYSQL_RES   *result;
MYSQL_ROW   row;
uint num_fields, i;
ulong *lengths;

/***  Ініціалізація  з'єднання.*/
if(!mysql_init(&mysql))
{
fprintf(stderr,   "Unable  to initialize   MYSQL   struct!\n");
exit (); }

/***  Підключення  до  бази  даних.*/
if(!mysql_real_connect(&mysql, "localhost", "root", "password","mysql", 0, NULL, 0)) {
fprintf(stderr, "%d: %s\n", mysql_errno(&mysql),mysql_error(&mysql));
exit(); }

/***  Передача  запиту  серверу.*/
if(!mysql_query(&mysql,"SELECT User, Host FROM user ORDER BY 1,2")) {

/***  Запит  завершився  невдачею!*/
fprintf(stderr, "%d: %s\n", mysql_errno(&mysql),mysql_error(&mysql));
}
else
{

/***  Занесення результатів в буфер, підрахунок числа полів.*/
result = mysql_store_result(&mysql);
num_fields = mysql_num_fields(result);
while(row = mysql_fetch_row(result))
{
lengths = mysql_fetch_lengths(result);
for(i = 0; i < num_fields; i++)
{
printf("[%.*s]",  (int) lengths[i], row[i] ? row[i] : "NULL") ;
printf("\n"); }

/***  Звільнення  буфера. */
mysql_free_result(result) ;
}

/*** Закриття з'єднання. */
mysql_close(&mysql);
}
```

Зупинимося на тому фрагменті програми, де результати запиту заносяться в буфер. В принципі, наперед відомо, що таблиця результатів містить два стовпці, але програма написана таким чином, що можуть оброблятися результати довільної інструкції SELECT. Визначивши за допомогою функції mysql_num_fields() число стовпців, ми дістаємо можливість в циклі пройти по кожному полю кожного запису. Запис є масив полів, нумерація яких починається з нуля. Якщо необхідно визначити типи полів, скористайтеся функцією mysql_fetch_fields(), mysql_fetch_field() або mysql_fetch_field_direct().

У циклі while з таблиці результатів послідовно витягуються записи. Функція mysql_fetch_lengths() визначає розмірність кожного стовпця. Це не та розмірність, яка вказана у визначенні стовпця, а реальна довжина рядка, що міститься у кожній клітинці. Функція mysql_fetch_lengths() надзвичайно зручна, оскільки за допомогою функції strlen() не можна визначати довжину двійкових рядків.

У циклі for функція printf() відображає значення кожної клітинки. Зверніть увагу на специфікацію * у рядку формату. Вона примушує функцію шукати в списку аргументів ціле число, задаюче розмірність стрічкового аргументу. Завдяки цьому пропадає необхідність завершувати кожен рядок символом NULL.

Запис, повертаний функцією mysql_fetch_row(), є масивом вказівників на рядки. У разі порожнього рядка елемент масиву посилається на рядок, який починається з символу NULL, а функція mysql_fetch_lengths() повідомляє про те, що довжина такого рядка рівна нулю. Якщо ж стовпець містить значення NULL, то в масиві знаходитиметься порожній вказівник. За допомогою оператора ? такому вказівнику ставиться у відповідність рядок "NULL".

4.3 Зміна даних

Запити на вставку або оновлення даних не повертають набори записів. Вони теж виконуються за допомогою функції mysql_query(), але викликати функцію mysql_store_result() немає необхідності. Якщо потрібно дізнатися число доданих або змінених записів, скористайтеся функцією mysql_affected_rows() (лістинг 4.3).

Лістинг 4.3 – Виконання запитів на вставку та оновлення даних

```
#include <stdio.h>
#include   <mysql/mysql.h>
int main(int argc,   char  *argv[]) {
MYSQL mysql;
MYSQL_RES   *result;
MYSQL_ROW  row;
uint i;
char query[4096];

/*** Дані, що вставляються.*/
const char *names[] = {
"Lech", "Vicky", "Carl", "Ricky", "Nicki", "Jeff"
"Betty", "Tina", "Joey" };
uint num rows = sizeof(names) /sizeof(char *);

/** * Запити.*/
const char *query_create =
"CREATE TABLE testapi (\
ID INT(ll) NOT NULL AUTO_INCREMENT \
Name VARCHAR(64) \
PRIMARY KEY(ID))";

const char *query_insert =
"INSERT INTO testapi  (Name) VALUES ('%s')";
const char *query_delete ="DELETE FROM testapi WHERE ID < 4";
const char *query_update ="UPDATE testapi SET Name - 'None' WHERE ID=5";
const char *query_drop = "DROP TABLE testapi";

/*** Ініціалізація з'єднання.*/
if (!mysql_init (&mysql))
{
fprintf(stderr, "Unable to initialize MYSQL struct!\n");
exit () ;
}
/*** Підключення до бази даних.*/
if(!mysql_real_connect(&mysql, "localhost", NULL, NULL, "test", 0, NULL, 0) ) {
fprintf(stderr, "%d: %s\n", mysql_errno(&mysql), mysql_error(&mysql));
exit();
}
```

```
        /*** Створення таблиці.*/
        if(mysql_query{&mysql, query_create))
        {

        /*** Запит завершився невдачею!*/
        fprintf(stderr, "Could not create table!\n%d: %s\n",mysql_errno(&mysql), mysql_error
(&mysql));
        mysql_close(&mysql);
        exit();
        }

        /*** Вставка записів.*/
        for(i=0; i<num_rows; i++)
        {
        sprintf (query, query_insert, names [i]) ;
        if (mysql_query(&mysql, query))
        {

        /*** Запит завершився невдачею!*/
        fprintf(stderr, "Could, not insert row!\n%s\n%d: %s\n", query, mysql_errno(&mysql),
mysql_error(&mysql);
        mysql_close(&mysql);
        exit ();
        }
        printf("%d row insert\n", mysql_affected_rows(&mysql));

        /*** Видалення записів.*/
        if {mysql_query (&mysql, query_delete) )
        {

        /*** Запит завершився невдачею!*/
        fprintf (stderr, "Could not delete rows ! \n%d: %s\n",mysql_errno(&mysql),
mysql_error(&mysql));
        mysql_close(&mysql);
        exit();
        }
        printf("%d rows deleted\n", mysql_affected_rows(&mysql) );
        /*** Оновлення записів.*/
        if(mysql_query(&mysql, query_update))
        {
        /*** Запит завершився невдачею!*/
        Fprintf(stderr, "Could not update rows ! \n%d: %s\n.", mysql_errno(&mysql),
mysql_error(&mysql));
        mysql_close(&mysql);
        exit(); }
        printf("%d rows updated\n",   mysql_affected_rows (&mysql) );

        /***  Видалення таблиці.*/
        if(mysql_query(&mysql, query_drop))
        {

        /***  Запит завершився невдачею!*/
        fprintf(stderr,  "Could not drop table!\n%d:  %s\n", mysql_errno(&mysql),
mysql_error(imysql));
        mysql_close(&mysql);
        exit();
        }
        /** *  Закриття з'єднання. */
        mysql_close(&mysql);
        }
```

Представлена програма створює таблицю і додає в неї записи. Далі відбувається видалення частини записів, оновлення запису і, нарешті, видалення всієї таблиці. Зверніть увагу на спосіб вставки записів. У циклі for завдяки функції sprintf() відбувається повторне використання однієї і тієї ж інструкції INSERT, в яку підставляються різні параметри. Ця проста методика не так ефективна, як підготовка однієї інструкції INSERT, що вставляє групу записів. Але для програми, яка приймає довільні призначені для користувача запити на додавання, оновлення і видалення записів, така методика виявиться вельми зручною.

5 ОБҐРУНТУВАННЯ ЕКОНОМІЧНОЇ ЕФЕКТИВНОСТІ

Метою даної частини дипломної роботи є здійснення економічних розрахунків, спрямованих на визначення економічної ефективності розробки програми, і прийняття рішення про її подальший розвиток і впровадження або ж недоцільність проведення відповідної розробки.

5.1 Визначення стадій технологічного процесу та загальної тривалості проведення розробки

Для визначення загальної тривалості проведення НДР доцільно дані витрат часу по окремих операціях технологічного процесу звести у таблицю 5.1.

Таблиця 5.1 – Середній час виконання НДР та стадії (операції) технологічного процесу

№ п/п	Назва операції	Виконавець	Середній час виконання операції, год.
1	Узгодження технічного завдання	Керівник проекту	12
2	Розробка алгоритмів	Технік	48
3	Розробка дизайну	Консультант	48
4	Написання програмного коду	Технік	112
5	Тестування програми	Консультант	104
	Разом	–	324

5.2 Визначення витрат на оплату праці

При розрахунку заробітної плати кількість робочих днів у місяці приймаємо в середньому 24,5 днів/міс.

Приймемо наступні тарифні ставки:

Керівник – 6 грн./год.

Консультант – 5 грн./год.

Технік – 4 грн./год.

Основна заробітна плата

$$З_{осн.}=Т_с*К_г, \quad (5.1)$$

де: $Т_с$ – тарифна ставка;

$К_г$ – кількість відпрацьованих годин.

$$З_{осн.}=12*6+160*5+152*4=1480,00 \text{ грн.}$$

Додаткова заробітна плата становить 10% основної. Отже,

$$З_{дод.}=0,1*З_{осн.}=0,1*1480,00=148,00 \text{ грн.}$$

Загальні витрати на оплату праці становлять $З_{осн.}+З_{дод.}$, тобто

$$В_{оп}=148,00+1480,00=1628,00 \text{ грн.}$$

Відрахування на соціальні витрати становлять в сумі 37,5% від витрат на оплату праці. Тому $В_{сз}=1628,00*0,375=610,50$ грн.

Таблиця 5.2 – Зведені розрахунки витрат на оплату праці

№ п/п	Категорія працівників	Основна заробітна плата, грн.			Додаткова заробітна плата	Нарахування на ФОН	Всього витрати на оплату праці, грн.
		Тарифна ставка, грн.	Кількість відпрацьованих годин	Фактична нарахована зарплата в грн.			
1	Керівник	6	12	72	9,20	–	–
2	Консультант	5	160	800	80,00	–	–
3	Технік	4	152	608	60,80	–	–
	Всього		320	1480	148,00	610,50	2960,50

5.3 Розрахунок матеріальних витрат

Матеріальні витрати визначаються як добуток кількості витрачених матеріалів та їх ціни. Тобто,

$$M_{вi}=q_i*p_i, \quad (5.2)$$

де: q_i – кількість витраченого матеріалу і-го виду,

p_i – ціна матеріалу і-го виду.

Розрахунки матеріальних витрат зведемо в таблицю 5.3

Таблиця 5.3 – Зведені розрахунки матеріальних витрат

№ п/п	Найменування	Од. виміру	Факт. витрачено	Ціна за одиницю, грн.	Загальна сума витрат, грн.
1	Папір	пачка	3	20	60
2	Компакт-диски	шт.	4	3	12
3	Картридж	шт.	2	300	600
4	Тонер	шт.	1	30	30
5	Хостинг	місяців	12	10	120
	Разом	–	–	–	822

5.4 Розрахунок витрат на електроенергію

Витрати на електроенергію одиниці обладнання визначається за формулою

$$З_e=W*T*S, \quad (5.3)$$

де: W – потужність споживання обладнання,

T – кількість годин роботи обладнання,

S – вартсть кловат-години.

Оскільки при розробці програми задіяний керівник, два консультанти і два техніки і кожен з них використовує в роботі комп'ютер, то загальні витрати на електроенергію складає:

$$З_е=0,5*324*0,24=38,88 \text{ грн.}$$

5.5 Розрахунок транспортних затрат

$$Т_в=З_{мв}*0,08...0,1 \qquad (5.4)$$

Транспортні витрати прогнозуються у розмірі 10 відсотків від загальної суми матеріальних затрат. В нашому випадку це становить:

$$822,00*0,1=82,20 \text{ грн.}$$

5.6 Розрахунок суми амортизаційних відрахувань

Метою відновлення основних фондів в процесі їх використання при виробництві, повинні проводитись амортизаційні відрахування.

Тобто відбувається процес перенесення вартості основних фондів на вартість новоствореної продукції з метою їх повного відновлення.

Комп'ютери та оргтехніка належать до четвертої групи основних фондів. Для цієї групи річна норма амортизації становить 60%.

Для визначення амортизаційних відрахувань застосуємо формулу:

$$A = \frac{Б_в * Н_а}{100\%}, \qquad (5.5)$$

де: А – амортизаційні відрахування за звітний період,

$Б_в$ – балансова вартість групи основних фондів на початок звітного періоду,

$Н_а$ – норма амортизації, %.

Отже, використовуючи в роботі 1 комп'ютер балансовою вартістю 4200 грн., і затративши на виготовлення НДР 324 год., встановимо, що амортизаційні відрахування становлять:

$$4200*0,05/150*324=453,60 \text{ грн.}$$

5.7 Обчислення накладних витрат

Накладні витрати пов'язані з обслуговуванням виробництва, утриманням управлінського апарату тощо.

Накладні витрати становлять від 20 до 60 відсотків від суми основної та додаткової зарплати працівників. Приймемо накладні витрати рівними 30 відсотків.

Отже, $Н_в=0,3*В_{оп}=0,3*1628,00=488,40$ грн.

5.8 Складання кошторису витрат та визначення собівартості НДР

Результати проведених розрахунків зведемо у таблицю 5.2.

Таблиця 5.4 – Кошторис витрат НДР

Зміст витрат	Сума, грн.	% від загальної суми
Витрати на оплату праці	1628,00	39,48
Відрахування на соціальні заходи	610,50	14,81
Матеріальні витрати	822,00	19,93
Витрати на електроенергію	38,88	0,94
Транспортні витрати	82,20	1,99
Амортизаційні відрахування	453,60	11,00
Накладні витрати	488,40	11,84
Собівартість	4123,58	100,00

Собівартість розраховується як сума всіх витрат. В таблиці це число наведене і становить $С_в$=4123,58 грн.

5.9 Розрахунок ціни НДР

Ціна НДР визначається за формулою:

$$Ц = \frac{Св + \left(1 + P_{рен}\right) + К * B_{ні}}{К}\left(1 + ПДВ\right), \quad (5.6)$$

де: $Р_{рен}$ – рівень рентабельності, 30%.

К – кількість замовлень, од.,

$В_{ні}$ – вартість носія інформації,

ПДВ – ставка податку на додану вартість, 20%.

Отже, ціна продукту становить

$Ц = 4123{,}58 * \left(1 + 0{,}2\right) * \left(1 + 0{,}3\right)$=6432,78 грн.

5.10 Визначення економічної ефективності і терміну окупності капітальних вкладень

Економічна ефективність ($E_р$) полягає у відношення результату виробництва до затрачених ресурсів:

$$E_р = \frac{П}{C_в}, \quad (5.7)$$

де: П – прибуток,

$C_в$ – собівартість.

Обчислимо прибуток, як різницю між ціною і собівартістю, тобто:

$$П=Ц-C_в=6432,78-4123,58=2309,20 \text{ грн.}$$

Отримаємо Ер=2309,20/4123,58=0,56.

Термін окупності розрахуємо термін окупності розробки за формулою:

$$T_р = \frac{1}{E_р}. \quad (5.8)$$

Отже, $T_р$=1/0,56=1,8 років.

Зведені техніко-економічні показники НДР показані в таблиці 5.5.

Таблиця 5.5 – Техніко-економічні показники НДР

№ п/п	Показник	Значення
1	Собівартість, грн.	4123,58
2	Плановий прибуток, грн.	2309,20
3	Ціна, грн.	6432,78
4	Економічна ефективність	0,56
5	Термін окупності, років	1,8

В ході розрахунків було визначено, що показник економічної ефективності становить 0,56, а також термін окупності – 1,8 року. Зважаючи на те, що нормативним показником терміну окупності є показник до 5 років, то можна зробити висновок, що розробка системи є в межах високої ефективності.

6 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

6.1 Предмет та зміст безпеки життєдіяльності

Середовище існування – оточуюче людину середовище, яке обумовлене сукупністю діючих на цей час факторів – природного, суспільного, матеріального, духовного та іншого характеру. Основними властивостями цих факторів є їх здатність впливати на діяльність людини, її здоров'я та нащадків. Характер впливу, що реалізується, може бути безпосереднім, побічним, негайним чи віддаленим.

Метою вивчення дисципліни «Безпека життєдіяльності» є:

− підготовка людини до повноцінного життя в суспільстві, що динамічно змінюється;

− формування загальних системних уявлень;

− формування знань з питань методичного забезпечення в галузі обґрунтування рішень безпеки і їх здійснення в практиці сільськогосподарського виробництва.

Об'єктом вивчення дисципліни "Безпека життєдіяльності" є людина. Основними її потребами є:

1) фізіологічні;
2) особиста безпека;
3) соціальні;
4) престиж;
5) духовні.

Головним завданням досягнення особистої безпеки є гармонізація цієї потреби з потребами суспільства і держави.

БЖД – це наукова дисципліна, що вивчає небезпеку і захист від неї.

Мета БЖД – досягнення безпеки людини в місці існування. Безпека людини визначається відсутністю виробничих і невиробничих аварій, стихійних і інших природних лих, небезпечних чинників, що викликають травми або різке погіршення здоров'я, шкідливих чинників, що викликають захворювання людини і котрі знижують його працездатність.

До предметів вивчення БЖД можна віднести фізіологічні і психологічні можливості людини з погляду БЖД, формування безпечних умов і їх оптимізації тощо.

Досягнення гармонізації на базі загальної безпеки дає змогу скласти основу до реалізації всього комплексу потреб людини і забезпечити стабілізацію її психічного стану за рахунок відчуття особистої захищеності індивіда і суспільства від загроз, створених навколишнім середовищем.

Безпека діяльності людини – це сукупність:

- властивостей навколишнього середовища, які не завдають шкоди людині в процесі її діяльності;
- якостей людини та заходів і засобів, які запобігають можливій шкоді її здоров'ю.

Виходячи з визначеної сукупності, основним завданням дисципліни є розробка системи, що забезпечує безпеку життєдіяльності людини, суспільства та держави.

Структура вивчення безпеки життєдіяльності пов'язана з логікою встановлення безпеки. В основу системи встановлення безпеки кладуться теоретичні основи дисципліни, які в формулюються у вигляді відповідних елементів, та передумови, які сформовані у вигляді теоретичних основ (чи їх елементів). Цей розділ забезпечує встановлення змісту і розв'язання проблем в безпеці життєдіяльності, в питаннях наявності небезпек, взаємодії їх з людиною, наслідки розвитку негараздів, формування стабільних комфортних умов.

6.2 Джерела електростатичного випромінювання

Вибір того чи іншого способу захисту від дії електромагнітних випромінювань залежить від робочого діапазону частот, характеру виконуваних робіт, напруженості та щільності потоку енергії ЕМП, необхідного ступеня захисту.

Електромагнітні випромінювання розрізняють за частотою коливань або довжиною хвилі. Найдовші хвилі – це коливання промислової або іншої звукової частоти, а також ультразвукові. Вони мають довжину хвилі понад 10 км (або частоту менш як 30 кГц), довгі і середні радіохвилі (від 10 км до 100 м або до 3 МГц) застосовують не тільки в радіотехніці, а й для плавлення металу, гартування деталей, сушіння деревини та ін. У промисловій електротермії для нагрівання діелектриків використовують також короткі радіохвилі (завдовжки 100 – 10 м або до 30 МГц), що, як і ультракороткі (10 – 1 м або до 300 МГц), належать до коливань ультрависокої частоти (УВЧ).

Електричне чи електростатичне поле виникає в результаті багатьох причин, в тому числі і через опромінення екрана потоком заряджених частинок, тому на електростатично заряджених екранах накопичується пил, який може викликати запалення шкіри, призвести до появи вугрів, погіршення зору і т.д.

У загальному випадку електричне поле ПК за своєю природою подібне до поля, що створюється кінескопами телевізорів за рахунок використання в ЕПТ високої напруги.

Якщо в моніторі не застосовуються спеціальні технічні вирішення (фільтри), що забезпечують послаблення зовнішнього поля, то потенціал накопиченого заряду досягає 10 – 30 кВ. Його можна відчути, піднісши руку до екрана. Тіло людини може зарядитися до напруги в декілька кіловольт.

Дані про електричне поле наведені в таблиці 6.1 (дослідження міністерства охорони здоров’я Німеччини – BGA).

Статичне електрмагнітне випромінювання за своїм шкіфдливим впливом прирівнюється до коливань промислової та звукової частоти, отже, і засоби захисту від нього аналогічні.

При промисловій частоті спеціальні заходи захисту від дії електричних полів доводиться застосовувати тільки під час обслуговування електроустановок напругою 330 – 500 кВ і вище. Тоді використовують спеціальні костюми і взуття, які дають можливість навідним зарядам стікати в землю без неприємних для людини відчуттів, а також екрануючі металеві козирки над робочими місцями (приводами роз'єднувачів та ін.).

Таблиця 6.1 – Максимальні значення напруженості електричного поля, виміряні на відстані 50 см від екранів найбільш розповсюджених типів моніторів (близько 400)

Смуга частот	Електричне поле, В/м	Норми BGA
5-1000 Гц	4,08	2500-177
10-150 кГц	4,08	87
150-300 кГц	0,48	87
0,3-30 МГц	0,0024	87-27,5
30-300 МГц	0,0024	27,5

Використовувати ці козирки і костюми (так звані індивідуальні екрануючі комплекти) обов'язково тільки в розподільних пристроях напругою 750 кВ, під час робіт на опорах ЛЕП – 330 – 750 кВ або ж при напругах понад 5 кВ/м, коли перебування у такому електричному полі повинно тривати більше за гігієнічно допустимий час (понад 3 год при 5 – 10 кВ/м, 1,5 год. при 10 – 15 кВ/м, 10 хв. при 15 – 20 кВ/м і 5 хв. при 20 – 25 кВ/м).

Тривале перебування на землі під ЛЕП теж шкідливе. Під крайньою фазою в середині прольоту на ЛЕП напругою 330 кВ напруга становить 6 кВ/м, а на ЛЕП-500 – 14 кВ/м. Тому під час польових робіт під ЛЕП напругою 330 кВ і вище треба враховувати цю обставину і краще використовувати трактори та інші машини з металевою кабіною або з встановленими зверху і з боків екранами, які виготовлені з металевої сітки.

Автомашини і трактори на пневматичних шинах заряджаються в електричному полі ЛЕП зарядами хоч і малого значення, але напругою, що становить кілька кіловольт. Дотик до них людини, яка стоїть на землі, не смертельний, але спричиняє болісний удар розрядним струмом, що може призвести до мимовільних рухів, а отже, і до механічних травм від дотику до рухомих частин та ін. Тому бажано не залишати машину під ЛЕП, якщо треба зупинитися, то до виходу з кабіни заземлити машину спеціальним заземлювачем (у вигляді гирі з штирем), прикріпленим до машини гнучким проводом. Заземлення може бути постійним у вигляді диска або сошника. Електроогорожі під ЛЕП 330 – 750 кВ краще взагалі не робити, бо в протяжних металевих частинах наводяться такі електрорушійні сили (ерс), що, наприклад, електроогорожа завдовжки 300 м навіть під ЛЕП напругою 220 кВ може при замиканні на опір 1000 Ом (людина) створити струм 10 мА, а на опір 500 Ом (корова) – 30 мА. Провід для виноградників, оскільки він не ізолюється спеціально від землі, порівняно безпечний, особливо при розташуванні перпендикулярно до траси ЛЕП і заземленні на кінцях.

Принтери теж вносять свою частку негативного впливу на здоров’я людини електростатичного поля – лазерні при роботі виділяють озон.

На думку спеціалістів, надлишковий озон допомагає розвитку алергії і прямо пов’язаний із появою синдрому недомагання.

У звичайних лазерних принтерів, як і ксероксів, основним елементом конструкції є електростатичний барабан, на який повинен подаватися

електричний заряд напругою у 7 кВ, що перетворює деякі молекули кисню в навколишньому повітрі в молекули озону.

Медики рекомендують не піддавати персонал протягом 8-годинного робочого дня дії озону, концентрація якого в повітрі перевищує 0,1 частини на мільйон. Такий вміст озону викликає відчуття запалення в очах і верхніх дихальних шляхах, а більш високий вміст може викликати нудоту, головний біль, кашель, а дія протягом півгодини концентрації озону порядку 50 частин на мільйон одиниць може призвести до смерті.

Рівень вмісту озону у принтерів чи ксероксів із забрудненим фільтром (а вони забруднюються регулярно) може досягати 0,5 частини на мільйон. Тому лазерні принтери рекомендується встановлювати ближче до вікна.

У сучасних лазерних принтерах цю проблему намагаються розв’язати двома шляхами: за рахунок зміни конструкції (заміна дротяного електризатора, розміщеного поряд із поверхнею барабана, на довгий тонкий ролик із електропровідної гуми, притиснутої до поверхні барабана); за рахунок розробки нових додаткових фільтрів для вже існуючих лазерних принтерів.

6.3 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.

6.4 Заходи щодо зменшення впливу електростатичного випромінювання

До числа заходів зменшення впливу на працівників елктромагнітних випромінювань належать: організаційні, інженерно-технічні та лікарсько-профілактичні. Організаційні заходи здійснюють органи санітарного нагляду. Вони проводять санітарний нагляд за об'єктами, в яких використовуються джерела електромагнітних випромінювань. Крім того, ще на стадії проектування об'єктів потребує забезпечення таке розташування джерел електромагнітного випромінювання , яке б зводило до мінімуму їх вплив на працюючих.

Інженерно-технічні заходи передбачають використання в умовах виробництва дистанційного керування апаратурою, що є джерелом випромінювання, екранування джерел випромінювання, застосування індивідуальних заходів захисту (халатів, комбінезонів із металізованої тканини, з виводом на заземлюючий пристрій). Для захисту очей доцільно використовувати захисні окуляри ЗП5-90. Скло окулярів вкрито

напівпровідниковим оловом, що послаблює інтенсивність електромагнітної енергії при світло-пропусканні не нижче 75%.

Взагалі, засоби індивідуального захисту необхідно використовувати лише тоді, коли інші захисні засоби неможливі чи недостатньо ефективні: при проходженні через зони опромінення підвищеної інтенсивності, при ремонтних і налагоджувальних роботах в аварійних ситуаціях, під час короткочасного контролю та при зміні інтенсивності опромінення. Такі засоби незручні в експлуатації, обмежують можливість виконання трудових операцій, погіршують гігієнічні умови.

У радіочастотному діапазоні засоби індивідуального захисту працюють за принципом екранування людини з використанням відбиття і поглинання електромагнітних променів. Для захисту тіла використовується одяг з металізованих тканин і рідіопоглинаючих матеріалів. Металізовану тканину роблять із бавовняних ниток з розміщеним всередині них тонким проводом, або з бавовняних чи капронових ниток, спірально обвитих металевим дротом. Така тканина, наче металева сітка, і при відстані між нитками до 0,5 мм ослаблює випромінювання не менше як на 20...30 дБ. При зшиванні деталей захисного одягу треба забезпечити контакт ізольованих проводів. Тому електрогерметизацію швів здійснюють електропровідними масами чи клеями, які забезпечують гальванічний контакт або збільшують ємкісний зв'язок неконтактуючих проводів.

Лікарсько-профілактичні заходи передбачають проведення систематичних медичних оглядів працівників, які перебувають у зоні дії електромагнітних променів, обмеження в часі перебування людей в зоні підвищеної інтенсивності електромагнітних випромінювань, видачу працюючим безкоштовного лікарсько-профілактичного харчування, перерви санітарно-оздоровчого характеру.

Загальні рекомендації-характеристики при роботі на ПК стосовно зменшення шкідливого впливу електростатичного поля.

1. Дотримання обмежень за медичними вказівками.

2. Уважне ставлення до характеристик дисплея.

3. Правильна організація робочого місця.

4. Раціональна організація робочого часу.

5. Надавати перевагу використанню дисплеїв з високою роздільною здатністю і раціональним розміром екрана (15 дюймів і більше). Не використовувати CGA, EGA, HGA, MGA монітори.

6. Обов'язково ставити на незахищений дисплей екранні поляризаційні фільтри (сіточні, плівочні, пластмасові, скляні), які зменшують шкідливе електромагнітне та ультрафіолетове випромінювання, роблять мерехтіння менш помітним, захищають дисплей від осідання зарядженого пилу, знижують електростатичний заряд, затримують рентгенівське випромінювання.

7. Загальний час роботи з ПК не повинен перевищувати 50 % усього робочого часу, тобто не більше 4 годин на день для дорослих (за кордоном для операторів ПК передбачено скорочений робочий день – 6 годин) і 2 години для дітей.

8. Не слід перевищувати темпи роботи порядку 10 тис. нажимів клавіш за годину (приблизно 1500 слів).

9. При роботі з ПК необхідно робити 15-хвилинні перерви через кожних 2 години, а при інтенсивній роботі – через 1 годину і навіть півгодини.

10. У приміщенні, де встановлено ПК, необхідно підтримувати відносну вологість повітря не нижче 40 % з метою зменшення шкідливого впливу електростатичного поля.

11. Рекомендується відсунути все обладнання ПК (і взагалі все, що включено в електричну розетку) на 60-90 см від робочого місця.

12. Не рекомендується стояти перед пристроєм, що працює (наприклад, принтером при виводі документів на друк, особливо перед лазерним).

7 ЕКОЛОГІЯ

7.1 Актуальність екологічної проблеми

Дипломна робота стосується розробки програмного забезпечення, а тому при його розробці та експлуатації розробленої програми передбачається використання персонального комп'ютера. Даний пристрій (ПК) чинить свій негативний вплив на довкілля. Адже комп'ютер є джерелом електромагнітного випромінювання. Крім того існують питання утилізації обчислювальної техніки.

Крім персональних комп'ютерів широко використовується також інша обчислювальна техніка. Сюди можна віднести різноманітні мобільні пристрої: комунікатори, стільникові телефони, ноутбуки, КПК, смартфони тощо. Дія їхня аналогічна до впливу ПК на довкілля. Тому проаналізуємо вплив електромагнітного випромінювання цих пристроїв та заходи, спрямовані на зменшення цього впливу.

7.2 Основні джерела забруднення, що створює технічний об'єкт

Проблема електромагнітного випромінювання від персонального комп'ютера достатньо гостра в силу декількох причин.

Комп'ютер має відразу два джерела випромінювання (монітор та системний блок). Користувач ПК практично позбавлений можливості працювати на відстані від комп'ютера та піддається його впливу досить тривалий час.

Крім того існує ще декілька вторинних факторів, котрі ускладнюють ситуацію. До них можна віднести роботу у тісному приміщенні та концентрація багатьох ПК в одному місці а також слід врахувати те, що вся обчислювальна техніка потребує підключення до силової мережі живлення.

Дані мережі (лінії електропередач – ЛЕП) також є джерелом електромагнітних випромінювань.

Найсильнішим джерелом електромагнітного випромінювання є мотнітор, особливо його бокові і задня стінки, оскільки вони не мають спеціального захисного покриття.

На нинішньому етапі розвитку науково-технічного прогресу людина вносить істотні зміни до природного магнітного поля, додаючи геофізичним чинникам нові напрями і різко підвищуючи інтенсивність своєї дії.

Негативна дія електромагнітних полів на людину і на ті або інші компоненти екосистем прямо пропорційне потужності поля і часу опромінювання. Несприятлива дія електромагнітного поля, що створюється ЛЕП, виявляється вже при напруженості поля, рівній 1000 В/м. У людини порушуються ендокринна система, обмінні процеси, функції головного і спинного мозку і ін.

Дія неіонізуючих електромагнітних випромінювань від радіотелевізійних і радіолокацій станцій на місце існування людини пов'язана з формуванням високочастотної енергії. Японськими ученими виявлено, що в районах, розташованих поблизу могутніх випромінюючих теле- і радіоантен помітно підвищується захворювання катарактою очей. Медико-біологічна негативна дія електромагнітних випромінювань зростає з підвищенням частоти, тобто із зменшенням довжини хвиль.

Неіонізуючі електромагнітні випромінювання радіодіапазону від радіотелевізійних засобів зв'язку, радіолокаторів і інших об'єктів приводять до значних порушень фізіологічних функцій людини і тварин. Шкідлива дія на людський організм невидимого, але дуже небезпечного електромагнітного забруднення навколишнього середовища йде набагато швидшими темпами, ніж прогрес в електроніці.

З моменту відкриття радіо пройшло вже більше 100 років, і по потужності радіовипромінювання Земля стала у багато разів яскравіша за

Сонце, але основна частка цієї потужності поки доводиться на порівняно низькі частоти, до яких людина адаптована. Тому поки не помітні особливо шкідливі масові наслідки роботи могутніх радіостанцій і могутніх телецентрів, хоча їх потужність складає десятки і навіть сотні кіловат. Набагато шкідливішим є високочастотне випромінювання сантиметрового діапазону.

Мобільний зв'язок знаходиться поки на самому початку цього діапазону, але поступово просувається все далі (GSM 1800,1900).

Безпосереднім джерелом випромінювання в мобільному телефоні є його штирьова антена. Решта всіх джерел випромінювання (сам передавач, гетеродини приймача, синтезатор частоти і інше) настільки малопотужна, що їх можна не брати до уваги.

НВЧ випромінювання безпосередньо нагріває організм (повна аналогія зі НВЧ піччю). Потік крові зменшує нагрів, але наприклад кришталик ока не омивається кров'ю і при значному нагріві - руйнується, каламутніє. Ці зміни як правило незворотні. Даний процес супроводжується різзю в очах і шумом в голові. Дія випромінювання на мозок людини значно менша, оскільки мозок екранований черепною коробкою і має розвинену кровоносну систему. Різні стандарти мають різну здібність до нагріву організму. Телефон стандарту GSM 900/1800 небезпечніше чим телефон стандарту NMT 450 оскільки частота випромінювання вища. Правда в NMT 450 використовуються великі потужності.

На щастя потужність НВЧ-випромінювання телефону невелика і до перегріву кришталика і мозку справа не доходить. Але телефон на відміну від НВЧ печі випромінює складний модульований сигнал, який несе в собі інформацію. Біологічно-інформаційні взаємодії вивчені недостатньо, достовірні результати досліджень у відкритому друці не публікуються і невідомі насьогодні.

Стандарти стільникового зв'язку розроблені на заході, там же виготовляються власне апарати. Вважається, що санітарні норми у них достатньо жорсткі і можна сподіватися, що за нас про все поклопоталися. Це не факт, хоч би з тієї причини, що старі радянські норми вважали шкідливим опромінювання починаючи з щільності потоку потужності 10 мкВт/см^2. Починаючи з цієї межі, обмежувалася тривалість робочого дня, призначалося молоко, доплата за шкідливість і так далі Після введення ринкових відносин з'явилося повідомлення, що мінімальна шкідлива щільність потоку потужність складає вже 100 мкВт/см^2, тобто всі ми стали рівно вдесятеро здоровіше і міцніше. Хотілося б в це вірити. Правда, це говорить і про те, що питання про шкідливу дію НВЧ випромінювання вивчене не так вже і добре. Про реальну випромінювану потужність мобільного телефону інформації украй мало, але існує стандарт, згідно якому ця потужність складає до 2 Вт (або 2000000 мкВт). При цьому незрозуміло, чи це середня потужність, чи імпульсна (короткочасна). Швидше за все, це саме середня потужність, а імпульсна потужність значно вища (будь-який виробник стільникової апаратури бореться за дальність зв'язку, а значить, збільшуватиме потужність до межі). На голову потрапляє приблизно 20% випромінюваній потужності, тобто близько 400000 мкВт. Для відповідності старим нормам (припускаємо, що вся ця потужність розмазується по освітленій стороні голови рівномірно!) поверхня освітленої сторони голови повинна бути не менше 40000 см^2 (квадрат 2*2 метри). По нових нормах поверхня освітленої сторони голови повинна бути не менше 4000 см^2 (квадрат приблизно 63*63 см). Адже реальне опромінювання нерівномірне, тому і щільність потоку потужності на окремих ділянках голови буде значно вища.

Всі ці достатньо наближені міркування проводилися в припущенні, що в телефоні використовується класична штирьова антена завдовжки приблизно в чверть довжини хвилі (з врахуванням покриття це зразково 70 мм). У сучасних апаратах антени прагнуть робити значно коротше. Але чим

коротше антена, тим більше її так звана добротність. Добротність визначає величину запасеної енергії і ця запасена енергія знаходиться в ближньому полі, тобто поблизу антени і не випромінюється. Тому голові дістається і потужність, що випромінює, і запасена (або реактивна) енергія. За рахунок поглинання частини запасеної енергії головою, наявність голови біля короткої антени декілька знижує її добротність і передавачу легко працювати.

7.3 Заходи з ліквідації і зменшенню забруднення, що створює об’єкт

Основний спосіб захисту населення від можливої шкідливої дії електромагнітних полів від ліній електропередачі – створення охоронних зон шириною від 15 до 30 м залежно від напруги ЛЕП. Дана міра вимагає відчуження великих територій і виключення їх з користування в деяких видах господарської діяльності.

Рівень напруженості електромагнітних полів знижують також за допомогою пристрою різних екранів, у тому числі і зелених насаджень, вибору геометричних параметрів ЛЕП, заземлення тросів і інших заходів.

У стадії розробки знаходяться проекти заміни повітряних ліній ЛЕП на кабельних і підземної прокладки високовольтних ліній.

Для захисту населення від неіонізуючих електромагнітних випромінювань, що створюються радіотелевізійними засобами зв'язку і радіолокаторами також використовується метод захисту відстанню. З цією метою влаштовують санітарно-захисну зону, розміри якої повинні забезпечити гранично допустимий рівень напруженості поля в населених місцях. Короткохвильові радіостанції великої потужності (понад 100 кВт) розміщують далеко від житлової забудови, поза межами населеного пункту.

Концепція нормування електромагнітних полів і випромінювань передбачає:

− вироблення єдиної системи нормативних значень гранично допустимих рівнів електромагнітних полів і випромінювань;

− захист природних ресурсів від втрат, обумовлених дією цих полів на різні компоненти природного середовища;

− запобігання значним функціональним порушенням екосистем в результаті прямої або непрямої дії полів на ті або інші компоненти цих систем.

Із засобів захисту при використанні мобільних паристроїв можна використовувати або екран (дротяну сітку), що відображає, або поглинаючий екран (сітка з резистивних провідників, наприклад нитки просочені вуглецем), або їх комбінацію.

Наведемо деякі заходи безпеки.

1. Розмова по мобільному телефону необхідно зробити коротким не з міркувань тарифного плану, а для свого здоров'я.

2.У машині НВЧ випромінювання перевідбивається від металевого кузова і значно посилюється його шкідливий вплив. Рекомендується використання зовнішньої антени.

3.В умовах нестійкого прийому потужність апарату автоматично підвищується до максимальної величини. Рекомендується або утриматися від тривалих переговорів або знайти місце із стійким прийомом.

4.Якщо у користувача мобільного пристрою є дача або заміський будинок, то якнайкращим виходом буде використання стаціонарної зовнішньої кругової (наприклад автомобільної) або направленої антени.

5.Немалу небезпеку представляють також ретранслятори провайдерів. Антена такого ретраслятора постійно випромінює достатньо могутній сигнал причому на всі боки. Як з цим боротися? Переселяйтеся або подалі від антени, або живіть в панельному будинку. Арматура панелей дещо

екранує житло. Допомагає металева сітка на вікнах. Розмір вічка сітки не більше 10 см.

6.Застосування комплектів Mini Hands Free зменшує опромінювання голови і перерозподіляє його на все тіло. Але дріт комплекту працює як перевипромінююча антена.

7. Зміна геометричних розмірів антени, вигин, кручення неминуче погіршує умови прийому і потужність передавача неминуче збільшується. Використовуйте тільки фірмові, рідні антени.

8.При виборі моделі телефону перевагу віддавайте апаратам із зовнішніми антенами і хорошою заявленою в характеристиках чутливістю.

Слід зауважити, що застосування найновіших моделей обчислювальної техніки – теж один із способів зменшити шкідливий вплив випромінювання. Адже кожне настпуне покоління техніки конструювалося у відповідності до більш жорстких вимог щодо зменшення негативного впливу на довкілля.

Питання утилізації комп'ютертної техніки передбачає використання технологій переробки пластмас, металів (в тому числі і дорогоцінних), акумуляторних батарей. В зв'язку з цим варто зазначити, що ці технології супроводжуються викидом шкідливих речовин. Тому дотримання всіх вимог технологічних процесів щодо переробки обчислювальної техніки в утиль – досить актуальна задача.

ВИСНОВКИ

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.

В даній дипломній роботі було проведено аналіз життєвого циклу ІС. Розширено проаналізований та формалізовано процес проектування ІС. Розглянуто та формалізовано створення ПС на основі шарової архітектури. ПС складається з декількох шарів в залежності від типу системи, загалом більшість ПС мають шар представлення, бізнес шар та шар доступу до даних. Кожен з шарів має складові частини, компоненти, реалізовуючи які ми отримуємо додаток.

Сформований механізм формування альтернативних архітектур, маючи набір патернів, шо реалізують компоненти архітектури програмного додатку, та комбінуючи їх можна створити множину альтернатив, що будуть відрізнятися реалізацією, але виконувати одну й туж задачу.

Розроблена математична модель оцінювання альтернатив шляхом використання методу аналізу ієрархій Сааті, в якому шляхом попарного порівняння альтернативних архітектур по деякому критерію створюється матриця попарних порівнянь, яка перетворюється в матрицю оцінок альтернатив по різним критеріям оцінювання.

Розроблений алгоритм та математична модель процесу прийняття рішень в якому визначивши ваги критеріїв, шляхом лінійної згортки оцінок альтернатив по критеріям і ваг критеріїв оцінювання виявляється найліпша архітектура для поставлених вимог якості.

В ході виконання роботи був розроблений проект програмного комплексу, що реалізує та частково автоматизує поставлені задачі. Даний програмний комплекс описаний в даній дипломній роботі.

ПЕРЕЛІК ДЖЕРЕЛ

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.

ДОДАТКИ

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ТЕРНОПІЛЬСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
ІМЕНІ ІВАНА ПУЛЮЯ

М А Т Е Р І А Л И

VII НАУКОВО-ТЕХНІЧНОЇ КОНФЕРЕНЦІЇ

«ІНФОРМАЦІЙНІ МОДЕЛІ, СИСТЕМИ ТА ТЕХНОЛОГІЇ »

11–12 грудня 2019 року

ТЕРНОПІЛЬ
2019

СЕКЦІЯ 2. **ІНФОРМАЦІЙНІ СИСТЕМИ ТА ТЕХНОЛОГІЇ**

УДК 004.415.5
М. Горалечко, С. Метохір
Тернопільський національний технічний університет імені Івана Пулюя, Україна

СТВОРЕННЯ МНОЖИНИ АЛЬТЕРНАТИВНИХ АРХІТЕКТУР ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

UDC 004.415.5
M. Goralechko, S. Metokhir
(Ternopil Ivan Puluj National Technical University, Ukraine)

DEVELOPMENT OF THE SET OF ALTERNATIVE SOFTWARE ARCHITECTURES

Для подання архітектури програмних систем була прийнята концепція, в якій функціональність програми розділена на шари. Корпорація Microsoft розробила технологію, яка базується на даній концепції [1]. Згідно з цією технологією для кожного шару розроблено набори компонентів (патернів), які реалізують функціональність даного шару. Патерни згруповані у категорії (модулі), що призначені для вирішення певних стандартизованих задач.

Кожний програмний додаток, який проектується, поділяється на логічні частини, які відповідають шарам. Визначивши категорії задач, які будуть розв'язуватись певним шаром, обирається деякий компонент з існуючого набору і, таким чином, будується каркас архітектури. Але для кожного модулю існує, як правило, декілька патернів, тому отримаємо множину альтернативних архітектур.

Для автоматизації цього процесу пропонується використати експертну технологію, де знання організовані у вигляді фрейму, зображеного на рисунку 1.

Рисунок 1. Структура фрейму бази знань експертної системи App – ім'я програми/програмного додатку; Layer – ім'я шару; Category – ім'я категорії/модулю задач; Pattern – ім'я патерну/шаблон

База даних архітектур (репозиторій патернів) використовується для формування альтернативних архітектур із стандартних патернів. Тут зберігаються правила побудови архітектури у відповідності до типу програмного додатка.

Адміністратор репозиторію патернів поділяє додатки на шари і визначає задачі, які розв'язуються на певних шарах. Архітектор заповнює фрейм-шаблон, в якому незаповненим залишається останній слот.

На основі пошуку в репозиторії патернів (шаблонів) знаходиться відповідний компонент, який поміщується в каркас архітектури, створюючи таким чином архітектуру додатку.

Основним фактором, що істотно вплинув на етапи проектування та реалізації програмного комплексу став чинник застосування системи для підтримки прийняття експертних рішень по архітектурі програмних додатків різних типів при достатньо великій кількості альтернатив для кожного типу.

Тому при проектуванні системи були враховані наступні функціональні вимоги [2]:

- зручність застосування системи для підтримки прийняття рішень на множині альтернативних архітектур експертами;

- забезпечення ефективної роботи адміністратора репозиторію патернів архітектур та архітектора програмних додатків;
- система має легко реорганізовуватись при розширенні кількості типів архітектур програмних додатків з відповідними шарами та патернами;
- доступ на модифікацію даних, які розміщені в репозиторії патернів архітектур програмних додатків, повинні мати особи, які мають відповідні повноваження.

Опишемо процес створення альтернативних архітектур. В ході проектування необхідно обирати відповідний до ТЗ тип архітектури додатку та обрати множину патернів для заповнення компонентів шарів (модулів), загалом для кожного компонента можна знайти декілька патернів, що виконують однакові задачі, але мають різну логічну, структурну чи функціональну реалізацію. Як приклад патерни шару представлення / компоненту UI :

- MVC (Model-view-controller) pattern [3].
- MVP (Model-View-Presenter) pattern [3].

За своєю функціональною метою вони реалізовують однаковий функціонал, але мають різну структурну та функціональну структуру. MVP є похідним, видозміненим патерном, відносно MVC.

Також як приклад можна розглянути патерни шару доступу до даних/компоненти доступу до даних:

- DAO (data access object) pattern [3].
- Пряма адресація [3].

За своєю функціональною метою обидва патерни реалізують доступ до даних в базі даних. Але DAO реалізовує декілька шарів абстракції за рахунок яких він є більш гнучким, захищеним та незалежним від типу бази. В свою чергу Пряма адресація реалізовую прямі запити до бази, що є швидшим методом отримання даних, але погано модернізованим та сильно залежним від типу та структури БД.

Припустимо, що обрано всі патерни архітектури однозначно, один компонент – один патерн. Таким чином комбінуючи компоненти ми отримуємо 4 альтернативні архітектури

Після створення множини альтернативних архітектур експертами проводиться попарне оцінювання сформованого масиву по різних критеріях якості. Під час цього оцінювання отримується матриця парних порівнянь по критеріях.

Було виконано порівняння альтернативних архітектур, після чого можемо зробити висновки. MVC більш широким та менш спеціалізованим патерном по відношенню до MVP. Таким чином архітектури з MVC краще модернізуються. Якщо порівняти патерни DAO та Пряму адресацію, то можна сказати, що Пряма адресація майже не піддається модернізації. В свою чергу DAO – є патерном, що розрахований на модернізацію.

Література

1. Руководство Microsoft по проектированию архитектуры приложений. 2-е издание. Microsoft, 2009. – 529 с.
2. 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.
3. Гамма, Э. Приемы объектно-ориентированного проектирования. Паттерны проектирования [Текст] / Э. Гамма, Р. Хелм, Р. Джонсон, Д. Влиссидес. – СПб. : Питер, 2010. – 366 с.