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

ПОЯСНЮВАЛЬНА ЗАПИСКА

до дипломного проекту (роботи)

магістр
(освітній рівень)

на тему: Засоби автоматизованого порівняння архітектур програмного забезпечення під час проектування комп'ютерних систем

Виконав: студент (ка) 6 курсу, групи СІм-61
напряму підготовки (спеціальності) 123
Комп'ютерна інженерія
(шифр і назва напряму підготовки, спеціальності)
(підпис) Горалечко М.В
(прізвище та ініціали)

Керівник (підпис) Лукенко А.М.
(прізвище та ініціали)

Нормоконтроль (підпис) Гиш Є.В.
(прізвище та ініціали)

Рецензент (підпис) Прийомак М.В
(прізвище та ініціали)

м. Тернопіль – 2019

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя
(повне найменування вищого навчального закладу)

Факультет Комп'ютерно-інформаційних систем та програмної інженерії
Кафедра Комп'ютерних систем та мереж
Освітній рівень Магістр
Напрям підготовки
(шифр і назва)
Спеціальність 123 „Комп'ютерна інженерія"
(шифр і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри КС
Оцухівська Г.М.
«30» 09 2019 р.

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЕКТ (РОБОТУ) СТУДЕНТУ

Горомечко Михайло Володимирович
(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) Засоби автоматизованого порівняння архітектур програмного забезпечення під час проектування комп'ютерних систем

Керівник проекту (роботи) Лупенко Анатолій Миколайович професор кафедри КС
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
Затверджені наказом по університету від «30» 08 2019 року № 4/7-767
2. Термін подання студентом проекту (роботи) 24.12.19
3. Вихідні дані до проекту (роботи) Засоби автоматизованого порівняння архітектур

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
Вступ
Розділ 1 Актуальність та обговорення проблеми
Розділ 2 Методи та засоби розв'язку задачі
Розділ 3 Практична реалізація системи
Розділ 4 Обгрунтування економічної ефективності
Розділ 5 Охорона праці та безпека в надзвичайних ситуаціях
Розділ 6 Екологія
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
Вступ
Актуальність і задачі дослідження
Процес проектування програмних систем
Структура системи
Діаграма ролі Архітектор в схемі
Діаграма ролі Експерт в системі
Діаграма класів та Активності

6. Консультанти розділів проекту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Обґрунт. екон. ефект	Кидрук Н. Б.		
Охорона праці	Осуховська Р. М.	21.11.19	
Екологія	Лосота О. М.		
Безпека надзвичайних ситуацій	Стадник		

7. Дата видачі завдання 30.09.19

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Термін виконання етапів проекту (роботи)	Примітка
1	Отримання завдання	20.09.19	Виконав
2	Аналіз завдання	30.09.19	Виконав
3	Написання розділу 1.	15.10.19	Виконав
4.	Написання розділу 2.	30.10.19	Виконав
5	Написання розділу 3	20.11.19	Виконав
6	Обґрунтування економічної ефективності	30.11.19	Виконав
7.	Охорона праці та безпека в надзвичайних ситуаціях.	1.12.19	Виконав
8	Екологія	12.12.19	Виконав
9	Оформлення пояснювальної записки	21.12.19	Виконав
10	Оформлення графічної частини	22.12.19	Виконав
11	Попередній захист	23.12.19.	Виконав
12	Захист	24.12.19	Виконав

Студент _____ (підпис) Горлечко М. В (прізвище та ініціали)

Керівник проекту (роботи) _____ (підпис) Луценко А. М. (прізвище та ініціали)

АНОТАЦІЯ

У роботі розглянуті питання проектування програмних систем, коли широко застосовується компонентна технологія, яка базується на вживанні компонентів повторного використання , 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.

Ключові слова: АЛЬТЕРНАТИВНІ АРХІТЕКТУРИ, ПАТЕРНИ ПРОЕКТУВАННЯ, ПРОГРАМНИЙ МОДУЛЬ, ЯКІСТЬ ПРОГРАМНИХ СИСТЕМ, МОДЕЛІ ЯКОСТІ, РОЗРОБКА ВИМОГ, МОДИФІКАЦІЯ ЖИТТЄВОГО ЦИКЛУ, ОЦІНЮВАННЯ ЯКОСТІ ПРОГРАМИХ СИСТЕМ, ІНФОРМАЦІЙНА СИСТЕМА.

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.

ANNOTATION

The paper discusses the design of software systems, when component technology is widely used, based on the use of reusable components, which are taken from earlier projects. The architecture in this technology is projected by choice, based on the requirements for the PS, the framework and filling it with the necessary components taken from the repository, or the Internet.

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 framework is a high-level abstraction of the PS project, and combines many interacting objects into some integrated environment. The extension of the concept of a component is a pattern (pattern) - an abstraction that includes a description of the interaction of a set of objects. In general cooperative activity, for which the roles of participants and their responsibilities are defined.

Since in the pattern repository, as a rule, several components that implement the same function, we get some set of alternative PS architectures. To select the most appropriate architecture variant, it is necessary to find estimates of alternatives with respect to quality criteria, given the constraints.

In practice, several methods for evaluating the software architecture are used. The best known of them are methods based on the development of usage and verification scenarios, satisfies this architecture option with a requirement for a certain quality criterion.

ЗМІСТ

ВСТУП

Актуальність теми. 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.

Мета дослідження: розгляд теоретичних та практичних засад технології побудови архітектури програмних додатків, формалізацію створення архітектур та оцінки їх.

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.

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.

Наукова новизна отриманих результатів. Наукова новизна полягає у вирішенні задачі забезпечення якості ПЗ на етапі проектування. При цьому було отримано такі результати:

– проаналізовано методику порівняльного оцінювання програмних архітектур;

– запропоновано програмну реалізацію оцінювання пріоритетів з використанням алгоритму простого вибору.

Практичне значення отриманих результатів. Всі розроблені методи можуть бути доведені до практичного впровадження у складі системи для розробника ПЗ. Така система дозволить реалізувати процес управління якістю ПЗ на етапі проектування архітектури шляхом розробки вимог якості, оцінювання та вибору найкращого з альтернативних проектів по визначеній множині критеріїв, можливості оперативної корекції оцінок при зміні вимог якості. А це дозволить підвищити якість проекту та зменшити ризик невідповідності виконаних проектів вимогам замовника.

Публікації. Основні положення роботи доповідались, розглядались та обговорювались на науковій конференції Тернопільського національного технічного університету. Результати дипломної роботи опубліковані у 2-х наукових працях, які є тезами студентської наукової конференції, яка проводились у ТНТУ.

Структура роботи. Робота складається з розрахунково-пояснювальної записки та графічної частини. Розрахунково-пояснювальна записка складається з вступу, 6 частин, висновків, переліку посилань та додатків. Обсяг роботи: розрахунково-пояснювальна записка – ____ арк. формату А4, 10 аркушів формату А1 графічної частини.

РОЗДІЛ 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.

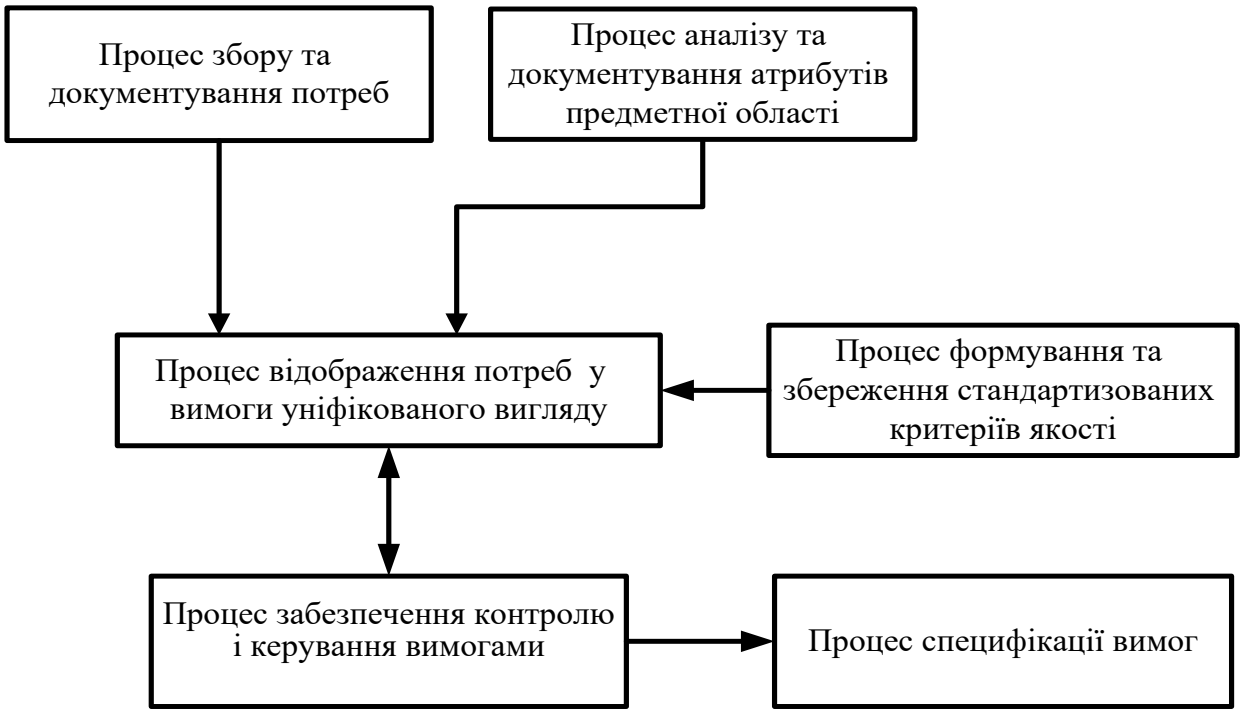


Рис. 1.1. Зв’язок процесів, які необхідно автоматизувати на етапі розробки вимог до ПС

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.

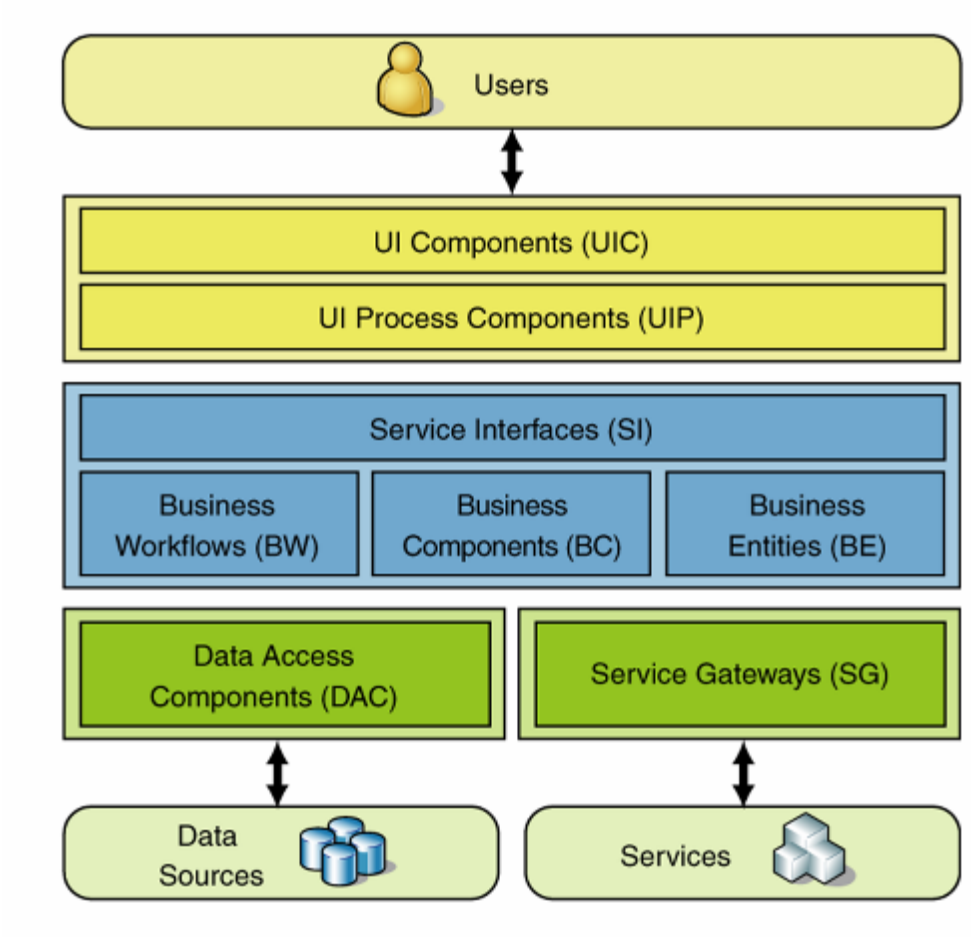


Рис. 1.2. Логічне подання багатошарової архітектури системи

– 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.

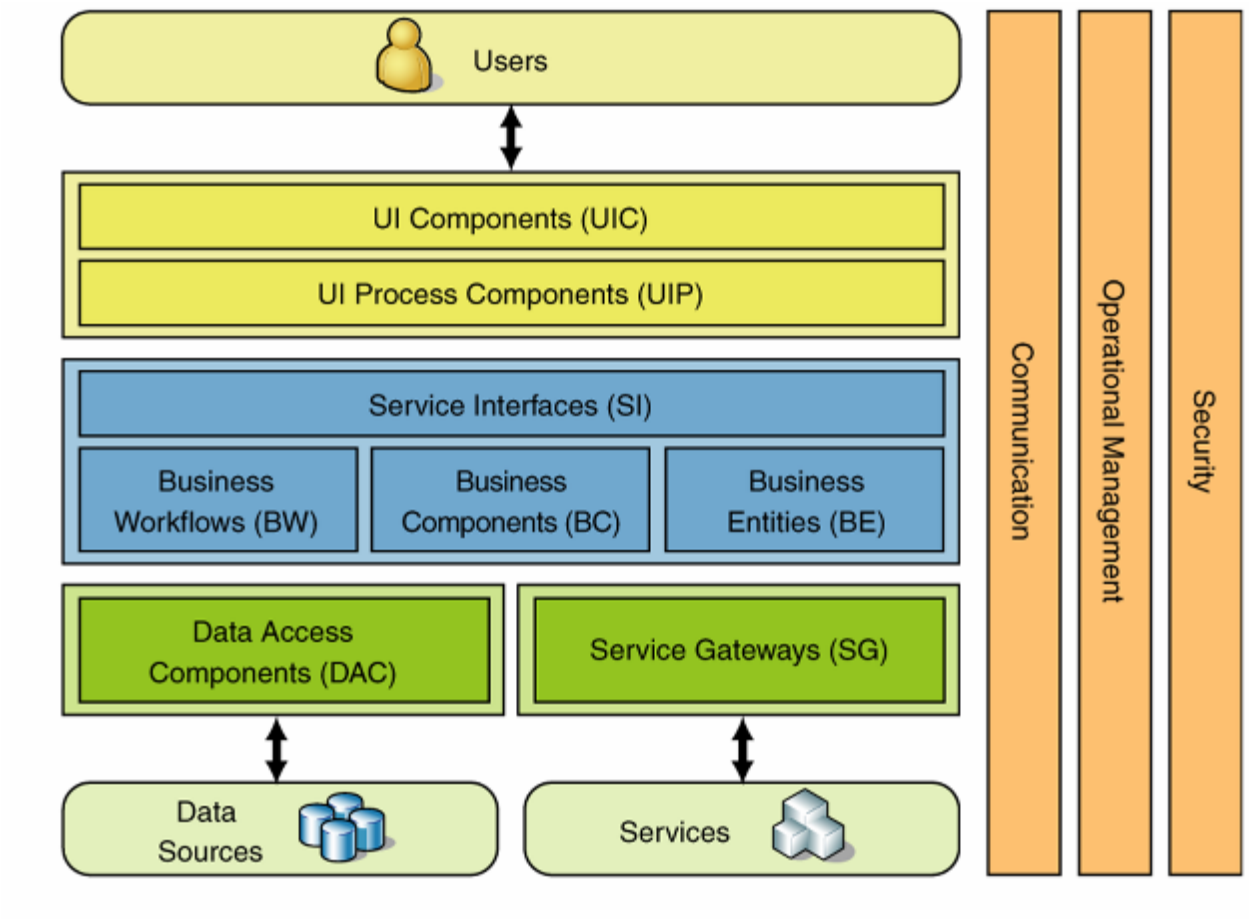


Рис. 1.3. Включення шару сервісів в додаток

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.

Рис. 1.4. Основні етапи ітеративного процесу проектування архітектури

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.

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 обосновних 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, итеративно 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 невзаімосвязанние 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 інструментірованіем. 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).

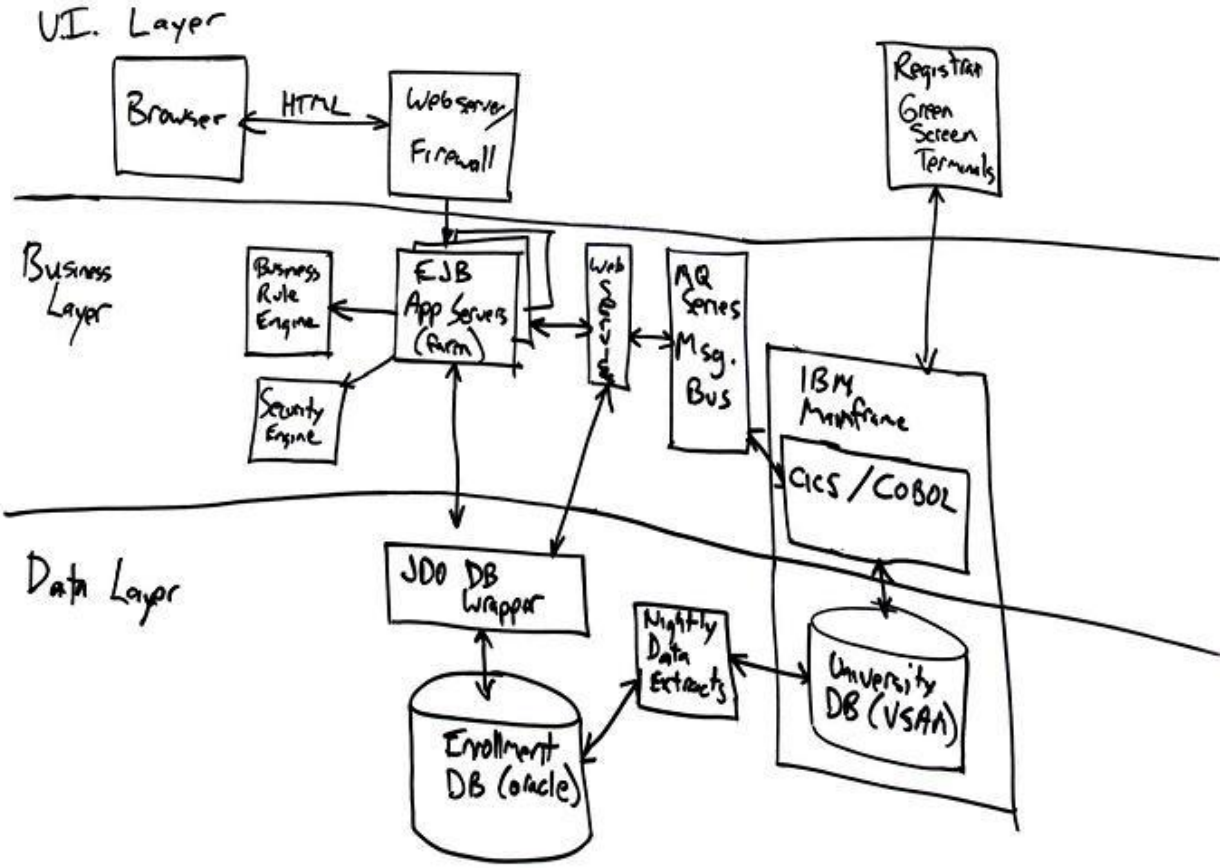


Рис. 1.5. Приклад графічного представлення дизайну веб-додатку в першому наближенні із зазначенням протоколів і методів аутентифікації, які передбачається використовувати

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 АТАМі 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/АТАМвибрані 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:

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

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 МАІ є обмежена кількість альтернатив, які можна оцінювати одночасно ($n<=7\pm2$), що викликано неузгодженістю елементів матриць парних порівнянь. Для вирішення цієї проблеми, в запропонована модифікація МАІ, 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 МАІ (ММАІ) до задачі вибору оптимальної архітектури програмних систем. Отримані відносні оцінки альтернатив в ММАІ 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.

РОЗДІЛ 2

МЕТОДИ ТА ЗАСОБИ РОЗВ'ЯЗКУ ЗАДАЧІ

2.1. Загальна концепція роботи системи

Для відображення програмної архітектури (ПА) була використана ідея, де усі функціональні вимоги дло програми відносяться до різних шарів. На основі цієї ідеї корпорація Microsoft розробила технологію для проектування ПА [7]. Відповідно до цієї технології кожен шар може містити деякі компоненти (шаблони, патерни), які реалізують функціональні вимоги свого шару. Патерни згруповані у модулі, які призначені для вирішення певних конкретних часткових задач.

Кожен проектований програмний засіб може бути поділений на логічні частини, які відповідають певним шарам згідно ідеї проектування архітектури ПЗ від Microsoft. Після визначення категорії задач, які будуть вирішуватись конкретним шаром, вибирається певний компонент з створеного ізаповнентого перед цим репозиторію і, таким чином, будується каркас архітектури шляхом компонування таких елементів. Справа в тому, що для кожного модулю в наявності є, як правило, кілька шаблонів, тому можна отримати декілька альтернативних програмних можливих пропозицій проектів архітектури проектованої системи , які реалізуютьл один і той же набір функціональних вимог.

Для автоматизації цього процесу пропонується використати експертну технологію, де знання організовані у вигляді фрейму, зображеного на рисунку 2.1.

App	Layer *i*	Category	Pattern *i*

Рис. 2.1. Структура фрейму бази знань експертної системи
App – ім’я програми/програмного засобу; Layer – ім’я шару; Category – ім’я категорії/модулю задач; Pattern – ім’я патерну/шаблон

База даних (БД) архітектурних патернів (репозиторій шаблонів) є джерелом для компонування альтернативних програмних можливих пропозицій проектів архітектури проектованої системи із типових шаблонів. Тут містяться правила компонування програмної архітектури у відповідності до предметної області для програмного продукту.

Адміністратор сховища патернів відповідно до шарового подання проектованих додатків вибирає задачі, які розв’язуються на кожному шарі. Архітектор запаковує слоти фрейму вибраними шаблонами проектування, один зі слотів залишається незаповнений для зв'язку з іншими шарами. З репозиторію патернів (шаблонів проектування) вибираються ті компоненти, котрі забезпечують фідповідну функціональну вимогу і поміщається у слот фрейма. Таким чином формується одне з альтернативних рішнь програмної архітектури.

Головним чинником, що суттєво впливає на етапи проектування та створення програмної системи, є застосування системи для підтримки прийняття рішень (ППР) експертами – архітекторами ПЗ різних типів при досить значній кількості альтернативних програмних архітектур для будь-якого типу. Тому при проектуванні системи були взяті до уваги такі функціональні вимоги до системи [*1*]:

– зручність користування системи для ППР для набору альтернативних програмних можливих пропозицій проектів архітектури проектованої системи Експертами;

– гарантування ефективної роботи Адміністратора сховища (репозиторію) архітектурних шаблонів, а також Архітектора проектованогопрограмного продукту;

– система повинна мати можливість перебудови архітектурних альтернатив при збільшенні кількості типів проектів ПЗ з власним розбиттям на шари та використання відповідних патернів, якуі відповідають конкретним функціональним вимогам;

– доступ до редагування та модифікації вмістимого репозиторію повинні мати авторизовані користувачі, на основі ролей, отриманих після автентифікації.

2.2. Метод створення множини архітектурних програмних рішень

Опишемо процес компонування множини програмних можливих пропозицій проектів архітектури проектованої системи. В ході проектування необхідно обирати відповідний до ТЗ тип архітектури застосунку та обрати множину шаблонів для заповнення компонентів шарів (модулів), загалом для кожного компонента можна знайти декілька шаблонів, що виконують однакові задачі, але мають різну логічну, структурну чи функціональну реалізацію. Як приклад патерни шару представлення / Компонента UI :

– MVC (Model-view-controller) pattern [*6*].

– MVP (Model-View-Presenter) pattern [*6*].

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 данних / Компоненти доступа до данних:

– DAO (data access object) pattern [*6*].

– Пряма адресація [*6*].

За своєю функціональною метою обидва патерни реалізують доступ до данних в БД. 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 данних, але погано модернізованим та сильно залежним від типу та структури БД.

Припустимо, що обрано всі патерни архітектури однозначно, один компонент – один патерн, за винятком компонентів: Компонети UI та Компоненти доступа до данних, де ми обрали по два патерни, які будуть реалізовувати альтернативи (див. рис. 2.2). Таким чином комбінуючи компоненти ми отримуємо 4 альтернативні програмні архітектурні рішення (див. рис.2.3).

2.3. Метод кількісної оцінки альтернативних програмних можливих пропозицій проектів архітектури проектованої системи

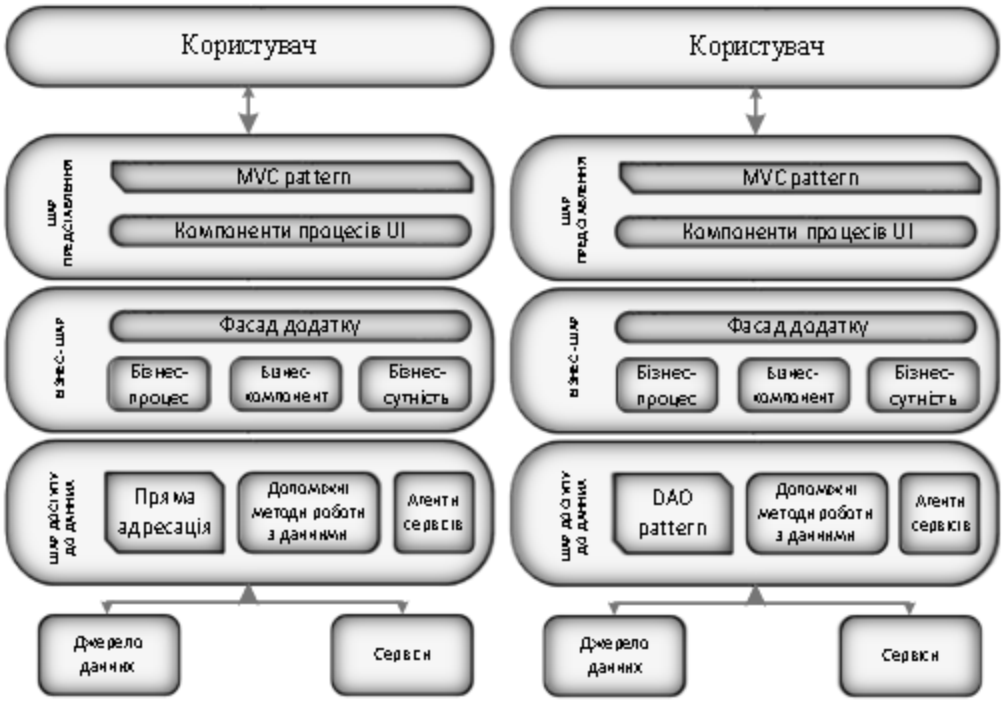
Після створення множини альтернативних програмних архітектур експертами проводиться попарне кількісної оцінки сформованого масиву за різними критеріями якості. Під час цього кількісної оцінки отримується матриця парних порівнянь.

Рис. 2.2. Відношення шаблонів до компонентів архітектури

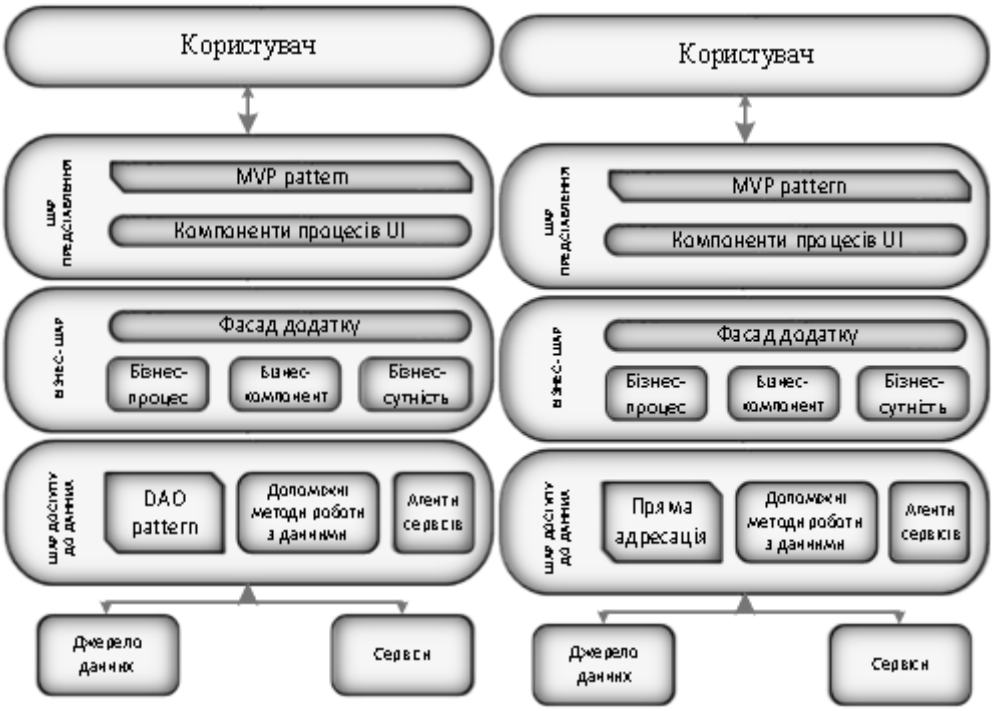
Як приклад множина альтернативних рішень з рисунка 2.3 альтернативи виглядать таким чином:

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

Заповнемо матрицю порівнянь за декількома критеріями, а саме модернізованості (таблиця 2.1) та швидкодії (таблиця 2.2). Розглянемо модернізованість MVC.



а) б)



в) г)

Рис. 2.3. Набір альтернативних програмних можливих пропозицій проектів архітектури проектованої системи :

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

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

Тут вона є 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.

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

Розглянемо Швидкодію MVC та MVP мають однакову порівняльну швидкодію, DAO – є патерном що реалізовує декілька шарів абстракції, за рахунок чого його швидкодія низча за Пряму адресацію, що безпосередньо працює з базою. Заповнюємо таблицю парних порівнянь (табл. 2.2).

Таблиця 2.1

Матриця парних порівнянь по модернізованості

	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

Матриця парних порівнянь по параметру Швидкодія

	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 не аналітичними для сприйняття людиною, тому їх необхідно перетворити в більш аналітичну форму, що дасть змогу проаналізувати переваги та недоліки окремих можливих пропозицій проектів архітектури проектованої системи.

Таким чином, використовуючи метод аналізу ієрархій Сааті [9], можна визначити оцінки архітектур, що будуть легко порівювальні, по різним критеріям.

Для здійснення цієї операції використовується формула (2.3):

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

де W_i – оцінка даної архітектури;

a_{ij} – значення оцінки в матриці парних порівнянь архітектур (відношення і-ї архітектури до j-й);

n – загальна кількість альтернативних програмних можливих пропозицій проектів архітектури проектованої системи .

Після виконання усіх математичних операцій з матрицею парних порівнянь отримується таблиця лінійних оцінок альтернативних рішень (табл.2.3).

Таблиця 2.3

Матриця критеріальних оцінок альтернативних рішень

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 переваги та недоліки архітектур за певними критеріями, але вона не дає відповіді, яка архітектура є найліпшою в поданному ТЗ. Тому необхідно вияснити пріорітети (ваги) окремих критеріїв та виконати приведення критеріальних оцінок до єдиної комплексної оцінки. Це можна виконати шляхом заповнення матриці парних порівнянь для критеріїв (табл. 2.4.), після чого, використовуючи метод аналізу ієрархій Сааті (2.4), знайти ваги окремих критеріїв (табл.2.5). Далі шляхом лінійної згортки (2.5) знайти комплексний критерій для архітектур (табл.2.6.).

Таблиця 2.4

Матриця парних порівнянь критеріїв кількісної оцінки

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 i-го критерія к j-тому) .

n – загальна кількість критеріїв.

Таблиця 2.5

Матриця ваг критеріїв кількісної оцінки

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

$$K_i = \sum_{i}^{n} Q_i W_i, \quad (2.5)$$

де K_i – оцінка даної архітектури;

n – загальна кількість критеріїв кількісної оцінки

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

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

Таблиця 2.6

Матриця комплексних оцінок альтернативних рішень

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

Таким чином виходячи з табл. 2.6. можна зробити висновок, що архітектура №2, має переваги над іншими. При цьому архітектура №4 не

сильно їй програє, в свою чергу архітектури 1 та 3 сильно їм програють. Саме за результатами лінійної згортки може працпрацювати модуль прийняття рішень, де найкраща оцінка означає найвищий пріорітет архітектурного рішення до застосування при проектуванні.

2.4. Опис функціональної структури системи

Програмний комплек складається з трьох програм:

– Програма кмопонування альтернативних програмних можливих пропозицій проектів архітектури проектованої системи та попарної експерної оцінки їх.

– Програма перегляду оцінок.

– Програма виставлення критеріальних пріорітетів та прийняття рішення.

В системі компонування альтернативних програмних можливих пропозицій проектів архітектури проектованої системи та попарної експерної оцінки їх наявні три ролі користувача [*1*]:

– адміністратор – додає у базу нові архітектури, шари, модулі та патерни та обслуговує базу даних архітектур і систему в цілому (в данній роботі не розглядається).

– архітектор – комбінує патерни, для вирішення класів специфікованих задач шляхом компонування наборів альтернативних програмних можливих пропозицій проектів архітектури проектованої системи .

– експерт – оцінює сформовані архітектором множини альтернативних програмних можливих пропозицій проектів архітектури проектованої системи них проектів програмних засобів за певними визначеними критеріями якості.

2.4.1. Діаграми прецедентів системи

Для моделювання системи ізабезпечення автоматизованого генерування програмного коду було використано графічне моделювання на мові UML. При розробці моделі об’єктів системи для віддображення сутностей предметної області використовувались діаграми класів, а для розробки функціональних вимог – діаграми прецедентів (Use case діаграми).

Функціональні можливості Архітектора забезпечуються використанням підсистеми створення альтернативних програмних можливих пропозицій проектів архітектури проектованої системи. Діаграма варіантів застосування для роботи Архітектора показана на рисунку 2.4.

Актор:

– Архітектор – ініціює та очолює процес створення програмних архітектур, обирає початкову архітектуру та патерни для реалізації функціоналу шарів ПЗ, описує завдання, приймає рішення стосовно програнмих архітектур.

Сценарії застосування:

– Створення альтернативних програмних можливих пропозицій проектів архітектури проектованої системи – базовий use case, що представляє основне завдання компоненту і викликає інші сценарії.

– Підключення до БД – вибір БД (репозиторію шаблонів) для подальшої роботи з нею.

– Вибір архітектури – обрання батьківської архітектури для побудови декількоч можливих пропозицій проектів архітектури проектованої системи програмного додатку.

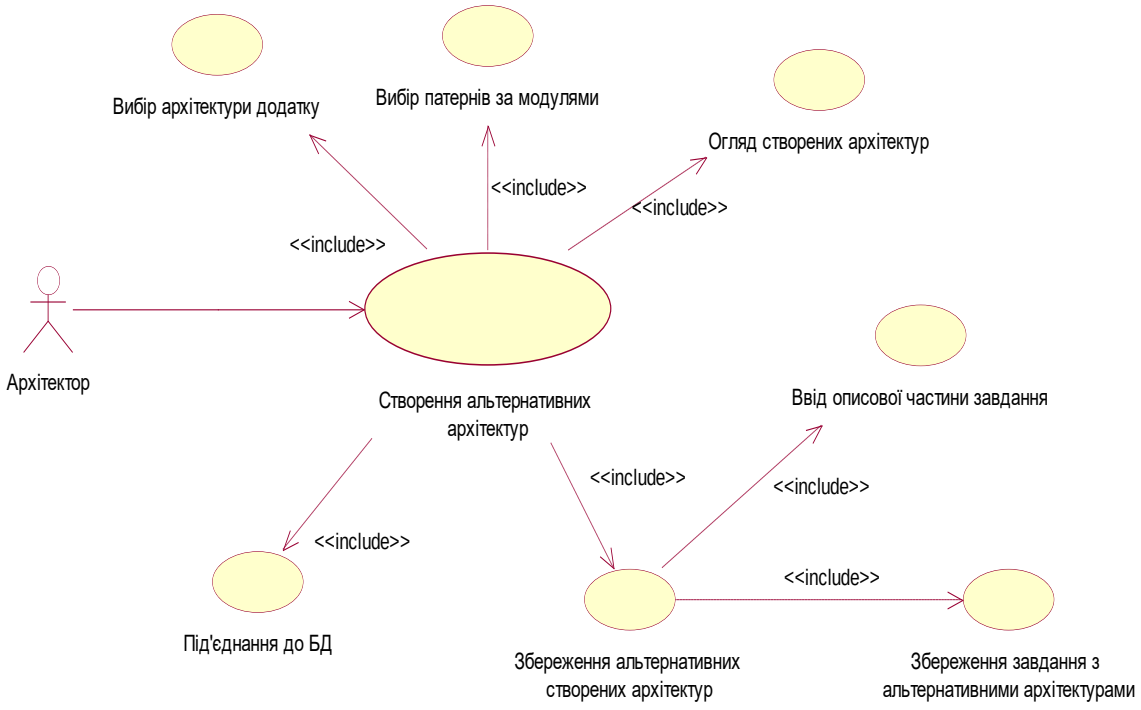


Рис. 2.4. Діаграма прецедентів для ролі Архітектора з компонування програмних архітектур

– Вибір шаблонів за модулями – вибір шаблонів для кожного модуля відповідного шару архітектури та компонування альтернативних рішень.

– Огляд створених архітектур – огляд архітектором створених декількох програмних можливих пропозицій проектів архітектури проектованої системи для ПЗ перед збереженням.

– Збереження альтернативних програмних можливих пропозицій проектів архітектури проектованої системи – здійснення збереження сформованих програмних рішень у вигляді завдання для подальшої роботи Експертів.

– Ввід детального опису завдання.

– Збереження завдання з альтернативними архітектурами додатків.

Функціональні можливості Експерта забезпечуються використанням підсистеми для кількісного оцінювання альтернативних програмних проектів архітектури. Use-case діаграма Експерта показана на рисунку 2.5.

Актор: Експерт – ініціює процес кількісної оцінки альтернативного рішення програмної архітектури ПЗ попарно одни з одним та встановює значення оцінок.

Сценарії використання відпоаідно до рисунку 2.5:

– Оцінка розроблених чи запропонованих програмних можливих пропозицій проектів архітектури проектованої системи – базовий use case, що формулює основну задачу частини системи і викликає інші сценарії.

– Вибір завдання для оцінки – вибір завдання для проведення процедури його кількісного оцінювання.

– Підключення до БД – вибір БД (репозиторію шаблонів архітектур) для подальшої роботи з нею.

– Зважена оцінка архітектур – компарація прорамних архітектур між собою попарно та отримання ї кількісної оцінки.

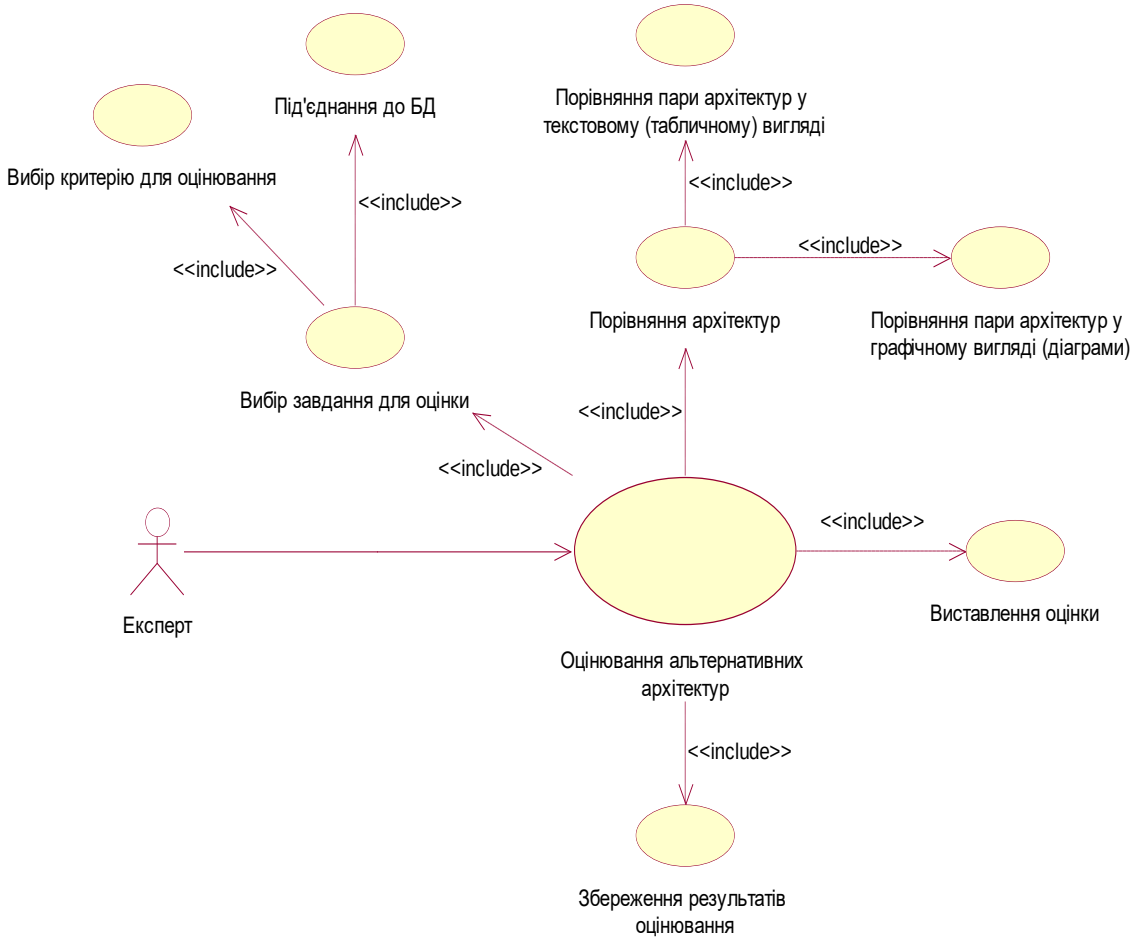


Рис. 2.5. Сценарій роботи Експерта з кількісної оцінки існуючих варіантів програмної архітектури

– Компарація пари архітектур у текстовому (табличному) та графічному (діаграми) вигляді.

– Ввід результатів порівняння архітекутрних рішень.

– Збереження результатів оцінки програмних архітектур.

Функціональні можливості Програми перегляду експертних оцінок. Use-case діаграма Програми перегляду експертних оцінок показана на рисунку2.6.

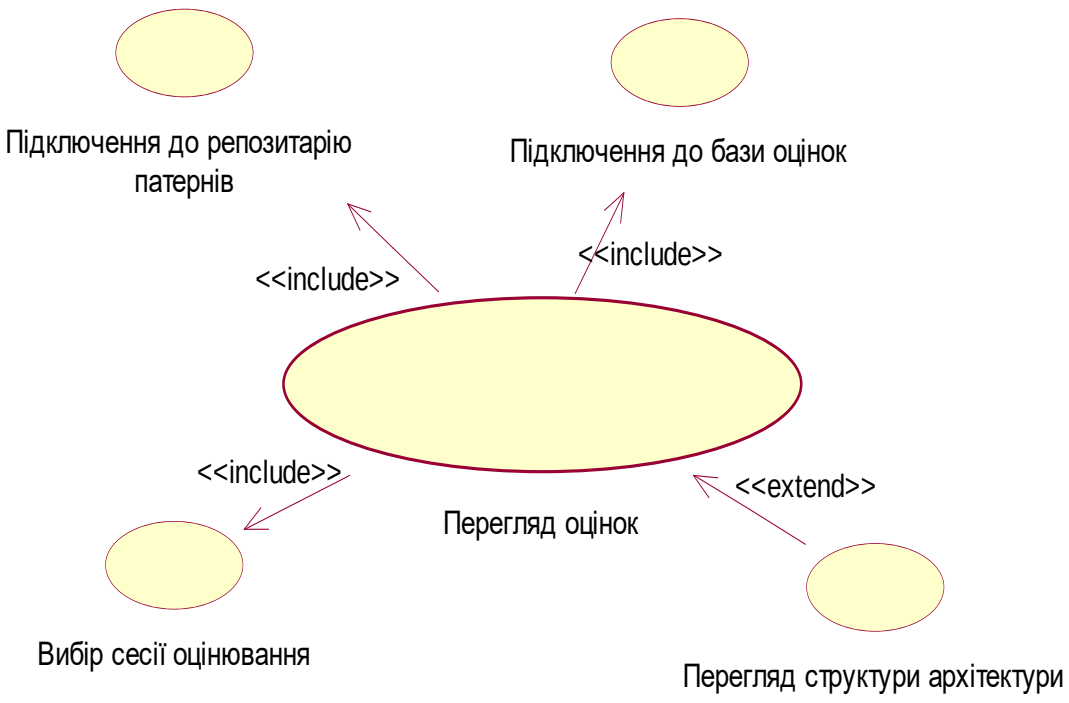


Рис. 2.6. Сценарій перегляду експертних оцінок

Варіанти застосування:

– Перегляд оцінок – основний сценарій використання, який формулює основну задачу програми і викликає інші сценарії.

– Підключення до репозитарію шаблонів – вибір БД (репозиторію шаблонів архітектур) для подальшої роботи з нею.

– Підключення до бази оцінок – вибір БД оцінок для подальшої роботи з нею.

– Перегляд структури архітектури – перегляд структури альтернативної архітектури.

– Вибір сесії кількісної оцінки – вибір сесії кількісної оцінки для візуалізації матриці парних порівнянь.

Функціональні можливості Програми виставлення критеріальних пріорітетів та прийняття рішення. Use-case діаграма Програми виставлення критеріальних пріорітетів та прийняття рішення показана на рисунку 2.7.

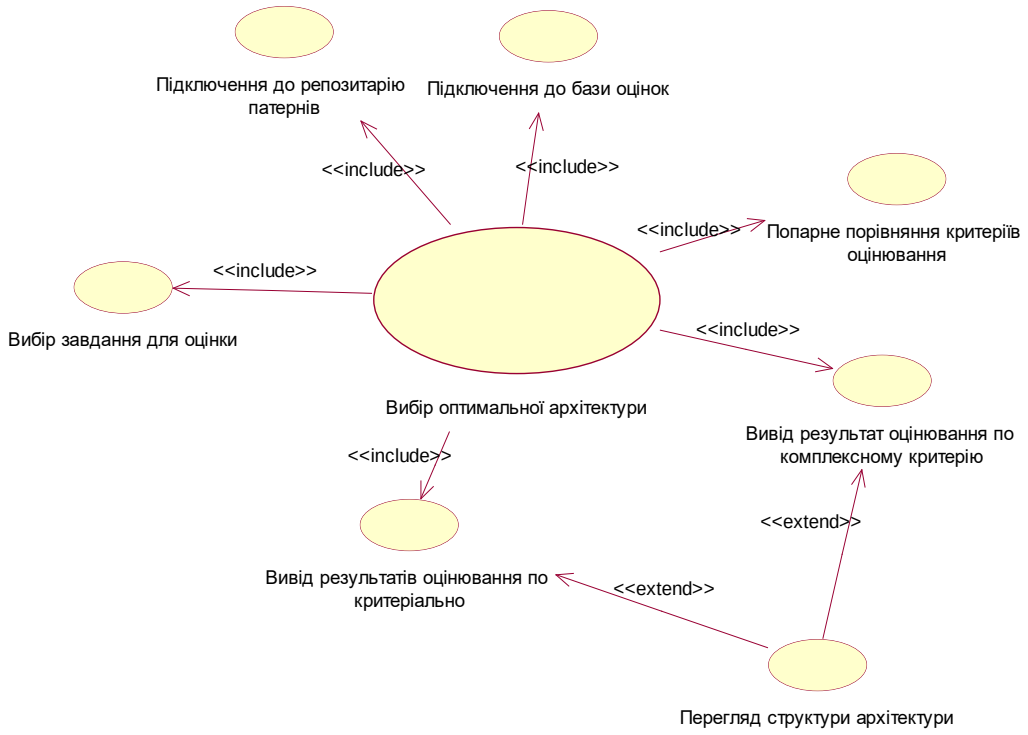


Рис. 2.7. Діаграма застосування програми виставлення критеріальних пріорітетів та прийняття рішення

Варіанти застосування:

– Вибір оптимальної архітектури – базовий use case, що описує головну задачу програми і визиває інші сценарії.

– Підключення до репозитарію шаблонів – вибір БД (репозиторію шаблонів архітектур) для подальшої роботи з нею.

– Підключення до бази оцінок – вибір БД оцінок для подальшої роботи з нею.

– Перегляд структури архітектури – перегляд структури альтернативної архітектури.

– Завдання для оцінки – вибір архітектурного рішення для порівняння з іншими архітектурами і вибіроптимальної.

– Попарне компарація критеріїв кількісної оцінки – виставлення оцінок парних порівнянь для критеріїв.

– Вивід результат кількісної оцінки по комплексному критерію.

– Вивід результатів кількісної оцінки по критеріально.

2.4.2. Архітектура системи

Розглянемо діаграми класів “Архітектора програмних систем” відповідно до трьох складових підсистем. Діаграма класів підсистеми управління репозиторієм шаблонів показана на рисунку 2.8.

Класи на діаграмі:

– Architecture – клас архітектури зі змінними і функціями для роботи з архітектурами програмних додатків.

– Layer – клас шар представляє собою набір даних і функцій для роботи з шарами програмного засобу.

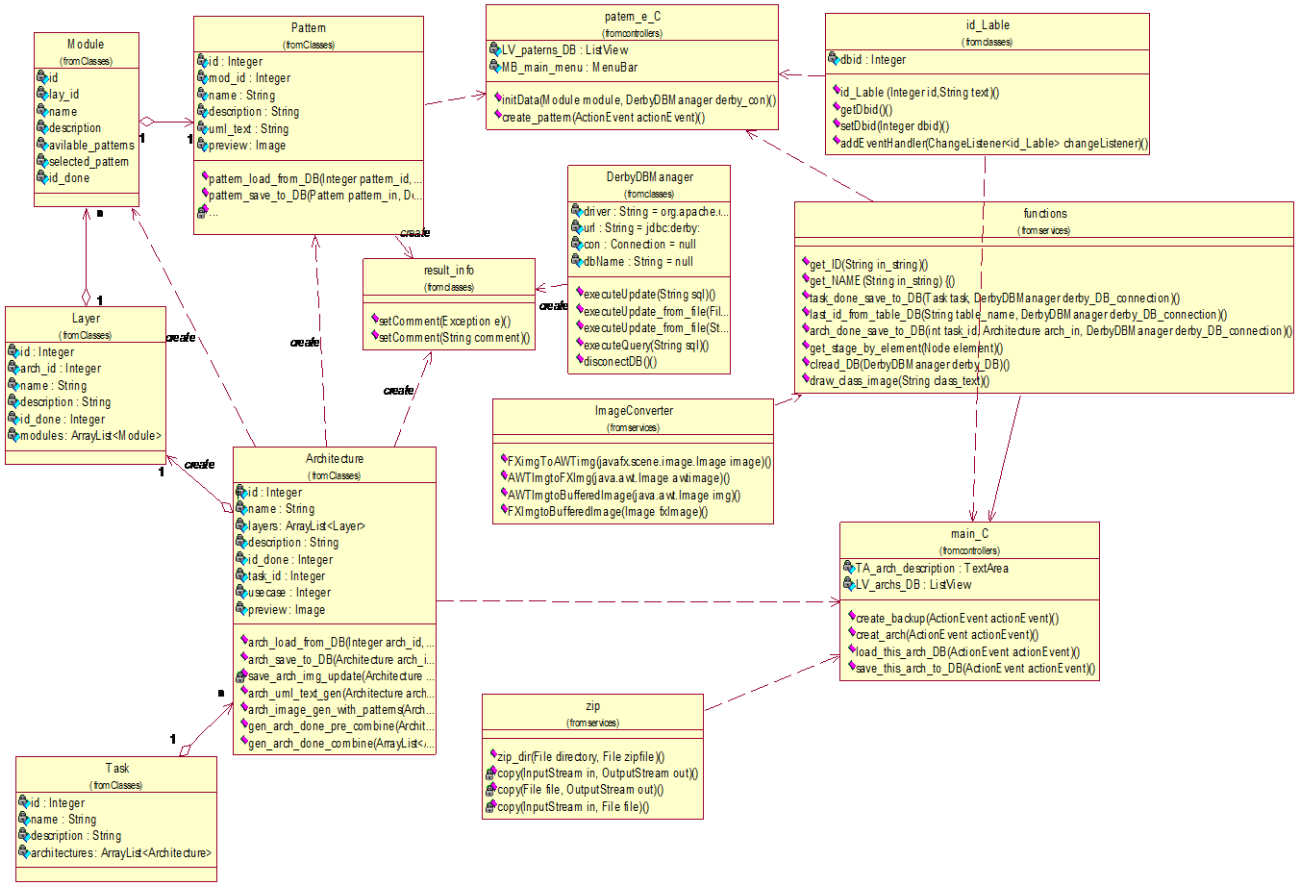


Рис. 2.8. Діаграма класів підсистеми управління репозиторієм шаблонів

– Module – клас модуль представляє собою комплект змінних і функцій для роботи з категоріями (модулями) шарів архітектури.

– Pattern – клас патерн представляє собою комплект даних і функцій для роботи з шаблонами (патернами) різних категорій.

– Task – клас задача, що використовується для збереженння множини побудованих програмних можливих пропозицій проектів архітектури проектованої системи-альтернатив для поточного типу додатка.

– result_info – клас, який повертає значення щодо здійснення операції (якщо відбулася помилка – повертається стек помилок).

– DerbyDBManager – клас роботи з БД архітектур.

– ImageConverter – клас-конвертор зображень архітектур.

– Zip – клас роботи з архівами, а саме архівація БД шаблонів архітектур.

– Functions – функціональний клас з різноманітними корисними функціями.

– id_Lable – клас на базі класу стандартного Lable, що містить в собі додаткову змінну, а саме ідентифікатор в БД.

– patern_e_C – клас-контролер редактора паттернів, який містить в собі функції кнопок і пунктів меню.

– main_C – клас-контролер головного вікна підсистеми управління репозиторієм шаблонів.

В таблиці 2.7 наведені специфікації основних функцій підсистеми.

Таблиця 2.7

Специфікації функцій підсистеми управління репозиторієм шаблонів

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 – Підключення до БД	Збереження задачі в базі даних

Продовження таблиці 2.7

functions	Integer last_id_from_table_DB (String table_name, DerbyDBManager derby_DB_connection)	table_name Ім’я таблиці derby_DB_connection Підключення до БД	Отримати максимальний (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 – Підключення 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 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, Dderby_DB_coection)	pattern_id –ID патерну який слід завантажити з БД derby_DB_coection- 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

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 БД
patern_e_C	create_pattern (ActionEvent actionEvent)	ActionEvent actionEvent	Створити патерн
	initData (DerbyDBManager derby_con)	derby_con – підключення до БД	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

Діаграма класів підсистеми створення альтернативних програмних можливих пропозицій проектів архітектури проектованої системи показана на рисунку 2.9.

Класи на діаграмі:

– Create_arch – клас, який інкапсулює основну бізнес-логіку підсистеми створення альтернативних програмних можливих пропозицій проектів архітектури проектованої системи .

– Function – допоміжний клас для роботи з БД, містить у собі функції роботи репозиторієм шаблонів.

– Arch_work – інкапсулює роботу з архітектурами програмних додатків.

– Gen_arch_done – допоміжний клас, генерує комбінації альтернативних програмних можливих пропозицій проектів архітектури проектованої системи .

Операції класів (функції):

– Вибір БД (репозиторію) – вібір БД (репозиторію), де зберігаються архітектури типів програмних додатків і патерни, над якими будуть виконуватись операції.

– Вибір архітектури застосунку – вибір батьківської архітектури для побудови альтернативних рішень.

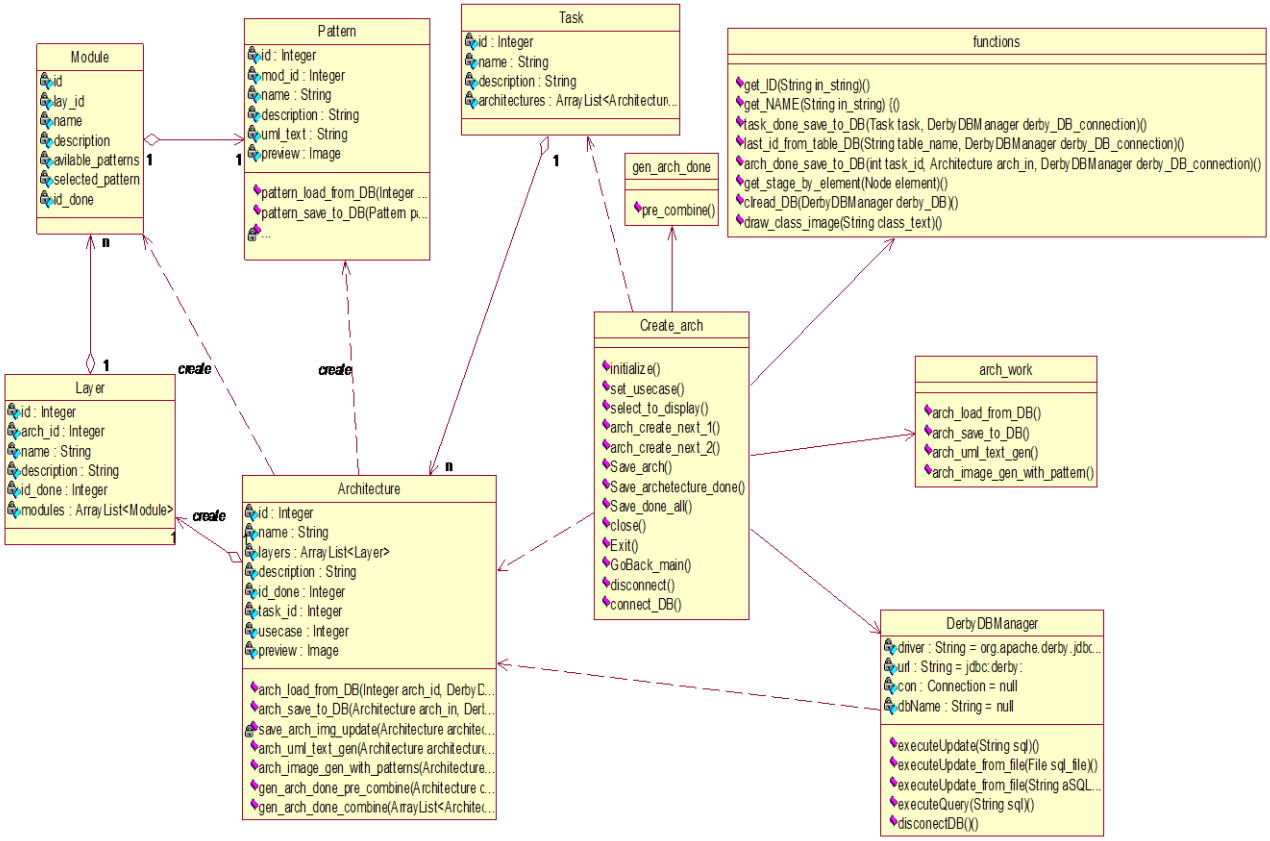


Рис. 2.9. Діаграма класів підсистеми створення альтернативних програмних можливих пропозицій проектів архітектури проектованої системи

– Вибір шаблонів по модулях – вибір шаблонів для кожного модуля з метою компонування альтернативних рішень.

– Візуалізація готових альтернативних програмних можливих пропозицій проектів архітектури проектованої системи.

– Створення завдання – ввід описової частини завдання на кількісної оцінки.

– Збереження альтернативних програмних можливих пропозицій проектів архітектури проектованої системи у БД.

– Компонування структури архітектури – компонування архітектури, з врахованням структури архітектури застосунку та насиченості її патернами.

– Генерація візуалізації архітектури та генерація альтернативних програмних можливих пропозицій проектів архітектури проектованої системи .

В таблиці 2.8 наведені специфікації основних функцій підсистеми створення альтернативних програмних можливих пропозицій проектів архітектури проектованої системи і підсистеми кількісної оцінки альтернативних проектів архітектури проектованої системи.

Таблиця 2.8

Специфікації функцій підсистеми створення альтернативних програмних можливих пропозицій проектів архітектури проектованої системи і підсистеми кількісної оцінки альтернативних проектів архітектури проектованої системи

Клас	Сигнатура функції	Опис параметрів	Опис функції
Create_arch	initialize (URL url, ResourceBundle rb)	URL url, ResourceBundle rb	Початковий опис інтерфейсу
	set_usecase (ActionEvent actionEvent)	ActionEvent	Вибір архітектури застосунку

Продовження таблиці 2.8

Create_arch	select_to_display (ActionEvent actionEvent)	ActionEvent	Відображення даних про архітектуру додатку
	Save_arch (ActionEvent actionEvent)	ActionEvent	Збереження альтернативних програмних архітектур
Get_arch_done	pre_combine (Architecture origin_arch, ArrayList<Module> modules_arr)	Architecture origin_arch – архітектура додатку , ArrayList<Module> modules_arr – масив заповнених модулів	Генерація альтернативних програмних можливих пропозицій проектів архітектури проектованої системи
Rating_arch	initialize (URL url, ResourceBundle rb)	URL url, ResourceBundle rb	Початковий опис інтерфейсу
	Start_rating		Вибір завдання
	task_description_view		Відображення опису завдання
Rating_arch	choice_task (ActionEvent actionEvent)	ActionEvent actionEvent	Завантаження архітектур за вибраним завданням
	draw_arch_im_text	ActionEvent actionEvent	Візуалізація архітектур для кількісної оцінки
	mark_done		Закінчення кількісної оцінки, 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

Діаграма класів підсистеми кількісної оцінки альтернативних проектів архітектури проектованої системи показана на рисунку 2.10.

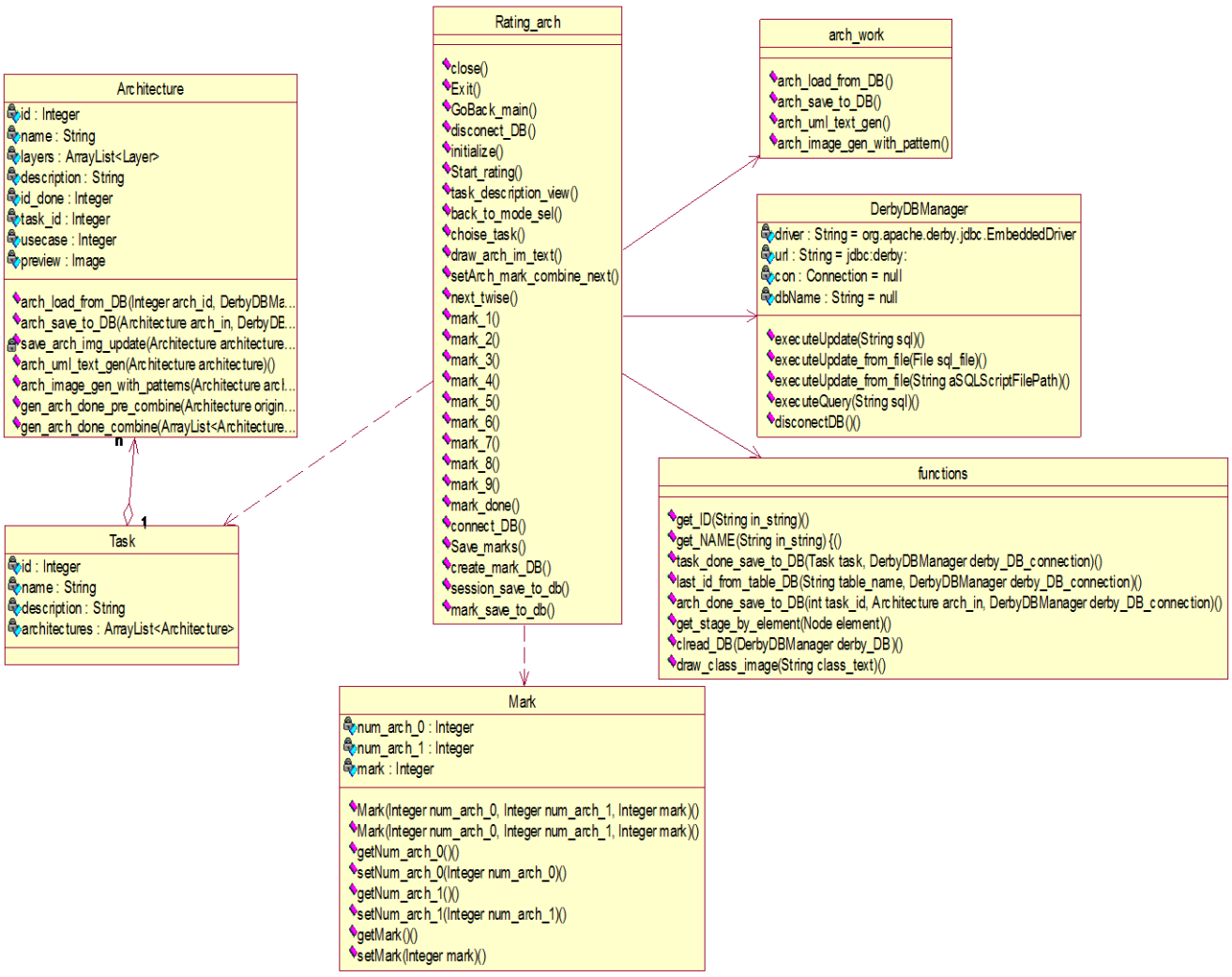


Рис. 2.10. Діаграма класів підсистеми кількісної оцінки альтернативних проектів архітектури проектованої системи

Класи на діаграмі:

– Rating_arch – клас, що інкапсулює головну бізнес-логіку системи, в якому проводиться кількісна оцінка альтернативних програмних пропозицій проектів архітектури проектованої системи.

– Arch_work – допоміжний (підрядний) клас, що містить в собі функції роботи з архітектурами.

Операції класів (функції):

– Вибір БД (репозиторію) – вібір БД (репозиторію), в якому зберігаються архітектури програмних додатків і патерни, над якими будуть виконуватись операції.

– Вибір завдання – вибір завдання на кількісної оцінки.

– Компонування текстового зображення архітектури.

– Виставлення оцінки – при виставленні оцінки матриця оцінок доповнюється та візуалізується нова пара архітектур для подальшого кількісної оцінки.

– Візуалізація матриці оцінок.

– Збереження оцінок.

– Компонування структури архітектури – компонування архітектури, враховуючи структуру архітектури застосунку та насиченість її патернами.

– Генерація візуалізації для архітектури.

Користуючись можливостями підсистеми створення альтернативних програмних можливих пропозицій проектів архітектури проектованої системи Архітектор ПС може виконувати дії, які показані на UML-діаграмі активності, що зображена на рисунку 2.11.

2.4.3. Діаграма «Створення альтернативних архітектур ПЗ»

Основні ролі діаграми

1. Архітектор – користувач, який виконує компонування множини альтернативних програмних пропозицій проектів архітектури проектованої системи.

2. Підсистема створення альтернативних пропозицій проектів архітектури проектованої системи – дії підсистеми системи “Архітектор програмних систем” по створенню альтернативних пропозицій для проектів архітектури розроблюваної системи .

3. База даних (БД) – репозиторій, де зберігаються архітектури і патерни додатків.

Дії (Activity):

1. Запит до БД – вибір репозиторію для подальшої роботи з ним.
2. Збір даних про архітектуру додатку – вибірка даних з БД про наявні архітектурні рішення для програмних додатків.
3. Список архітектур додатку – вивід списку варіантів архітектур розроблюваних додатків.
4. Вибір архітектури застосунку – вибір потрібної архітектури програмного засобу Архітектором.
5. Запит структури архітектури вибраної архітектури застосунку та шаблонів до неї.
6. Здійснення запиту – виконання попереднього запиту репозиторієм.
7. Вивід результатів – вивід результату запиту щодо детальної структури архітектури застосунку та шаблонів до неї.
8. Вибір шаблонів по модулям архітектури – вибір потрібних шаблонів по модулям та шарам архітектури програмного засобу.
9. Компонування альтернативних програмних можливих пропозицій проектів архітектури проектованої системи .
10. Перегляд створених архітектур Архітектором.
11. Компонування запиту на введення даних про завдання.
12. Введення даних про завдання.
13. Збереження завдання та архітектур до нього.
14. Збереження альтернативних програмних можливих пропозицій проектів архітектури проектованої системи у БД (репозиторії шаблонів).

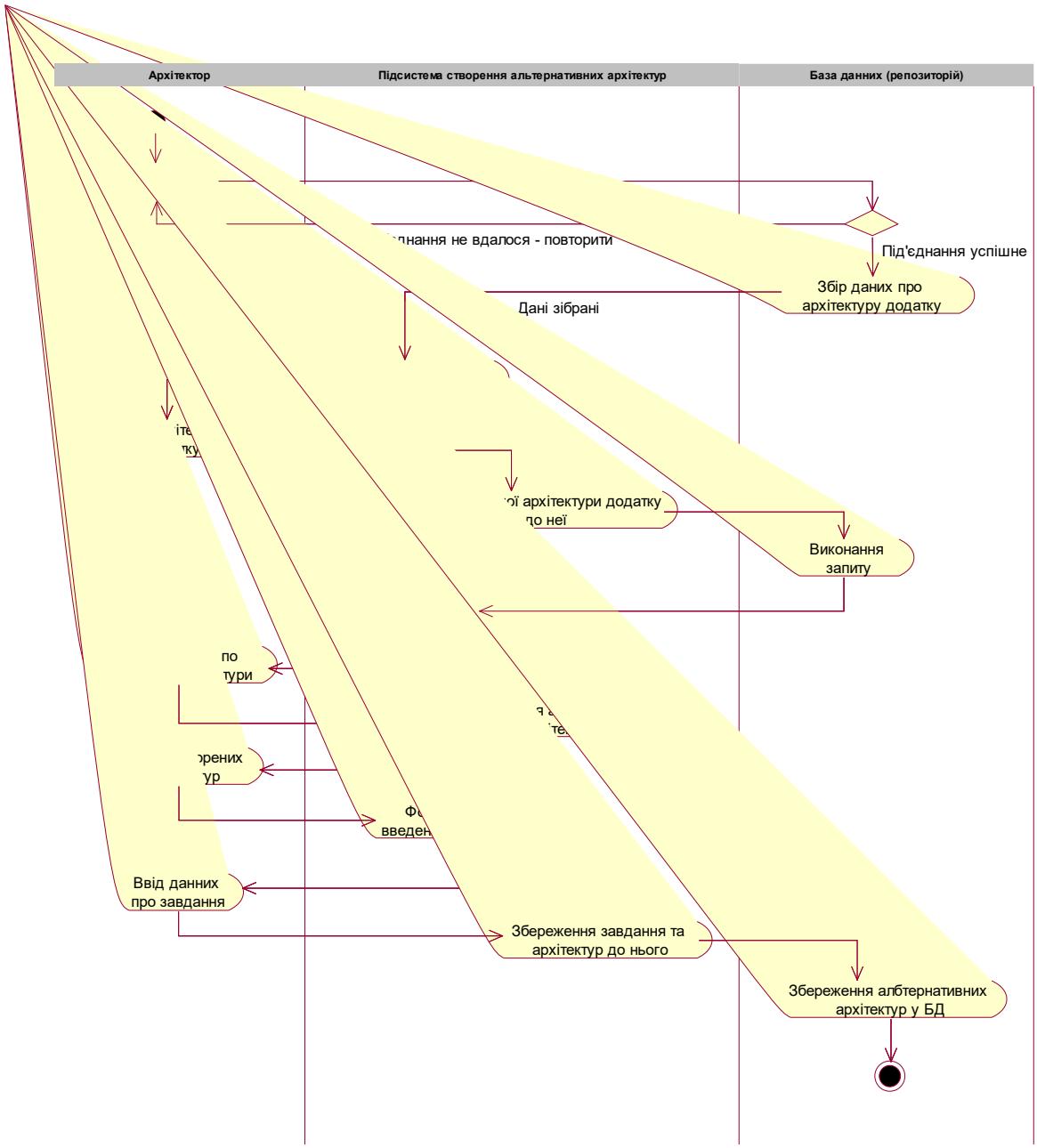


Рис. 2.11. Діаграма активності програмного Архітектора

Користуючись можливостями підсистеми кількісної оцінки альтернативних проектів архітектури проектованої системи, Експерт архітектур може виконувати дії, які показані на діаграмі активності, що зображена на рисунку 2.12.

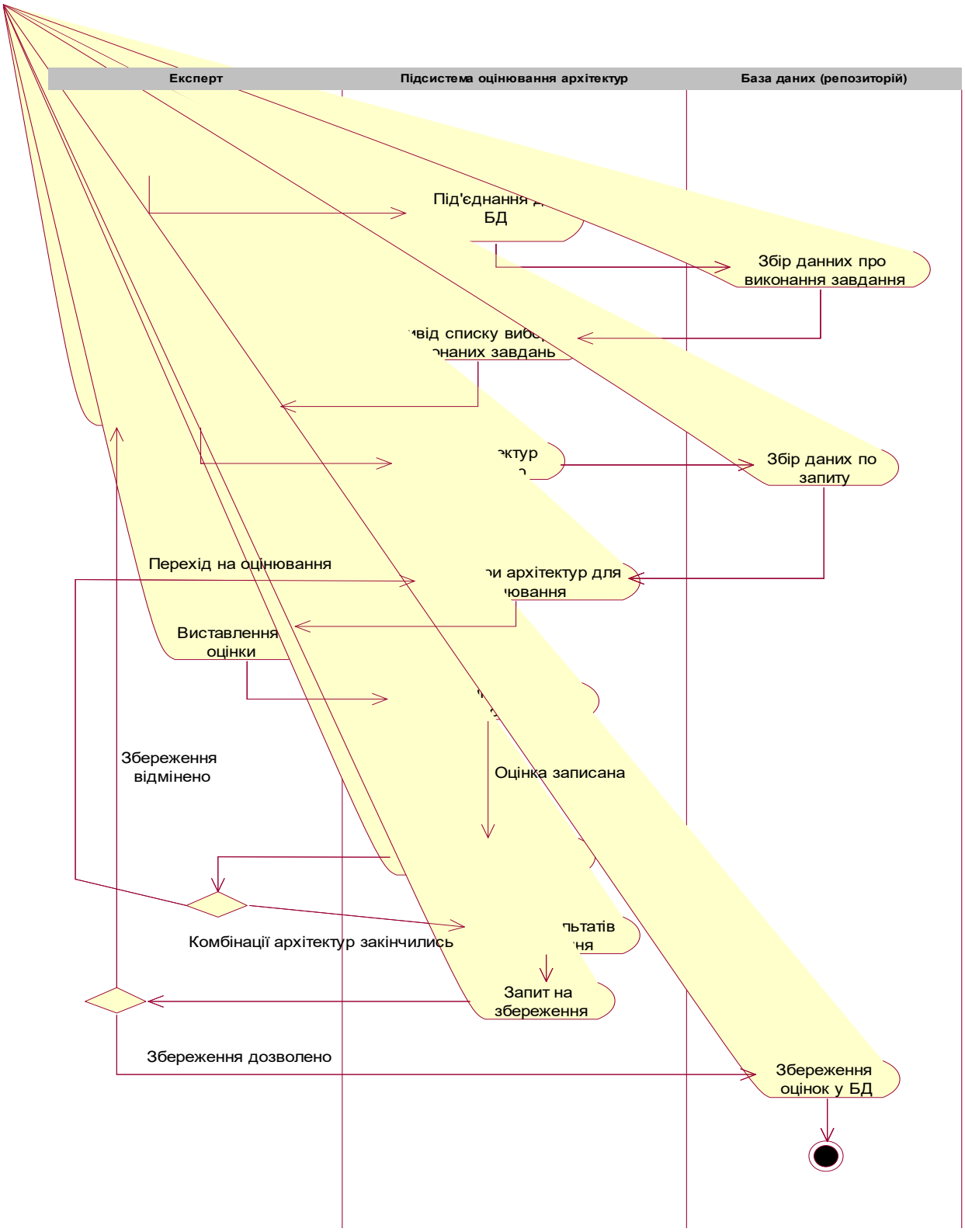


Рис. 2.12 – Діаграма активності Експерта по оцінюванню альтернативних програмних можливих пропозицій проектів архітектури проектованої системи додатку

2.4.4. Опис діаграми активності «Кількісної оцінки альтернативних програмних можливих пропозицій проектів архітектури проектованої системи »

Доріжки відповідальності:

– Експерт – дії користувача, котрий має оцінювати альтернативні програмні архітектурні рішення .

– Підсистема кількісної оцінки альтернативних проектів архітектури проектованої системи – дії підсистеми по оцінюванню архітектур.

– БД (репозиторій) – репозиторій, де зберігаються архітектури, патерни.

Дії (Activity):

1. Вибір БД – вибір репозиторію архітектур для подальшої роботи з ним.

2. Підключення до БД – підключення до БД (репозиторію шаблонів).

3. Збір даних про здійснення завдання – збір інформації про виконувані завдання по оцінюванню відповідних наборів архітектур.

4. Вивід для вибору виконаних завдань – вивід списку завдань для вибору.

5. Вибір завдання для проведення оцінки – аналіз інформації по завданню та вибір завдання для проведення кількісної оцінки.

6. Запит архітектур по завданню – компонування запиту відповідно завданню щодо множини архітектур, яка генерується варіантами шаблонів для типу додатка.

7. Збір даних по запиту – збір даних по запиту у репозиторії шаблонів.

8. Вивід пари архітектур для кількісної оцінки – вивід вікна кількісної оцінки альтернативних пар архітектур програмного засобу.

9. Виставлення оцінки – аналіз пари архітектур згідно обраного попередньо критерія та виставлення оцінки Експертом.

10. Заповнення матриці результатів кількісної оцінки – заповнення комірки таблиці попарного порівняння множини альтернативних програмних можливих пропозицій проектів архітектури проектованої системи оцінкою Експерта.

11. Перехід на наступну пару архітектур – вивід наступної пари архітектур для подальшого кількісної оцінки.

12. Вивід результатів кількісної оцінки – по закінченню кількісної оцінки виводиться інформативне вікно з матрицею порівняльних оцінок поточної множини альтернативних програмних можливих пропозицій проектів архітектури проектованої системи програмного засобу.

13. Запит на збереження – запит на збереження матриці порівняльних оцінок поточної сесії кількісної оцінки згідно обраного критерію.

14. Збереження оцінок у БД – збереження таблиці оцінок поточної сесії кількісної оцінки у відповідних таблицях БД.

Користуючись можливостями програми перегляду експертних оцінок, користувач може виконувати дії, які показані на діаграмі активності, що зображена на рисунку 2.13.

2.4.5. Опис діаграми активності «Програма перегляду експертних оцінок»

Ролі у діаграмі:

– Користувач – дії користувача, котрий має переглянути екпертні оцінки альтернативних програмних можливих пропозицій проектів архітектури проектованої системи .

– Система перегляду оцінок – система реалізовуе інтефейс роботи з базою оцінок, віалізовує архітектури та оцінки.

– Репозиторій шаблонів та архітектур – репозиторій, де зберігаються архітектури, патерни.

– База оцінок – база данних що здерігає сесії кількісної оцінки та оцінки

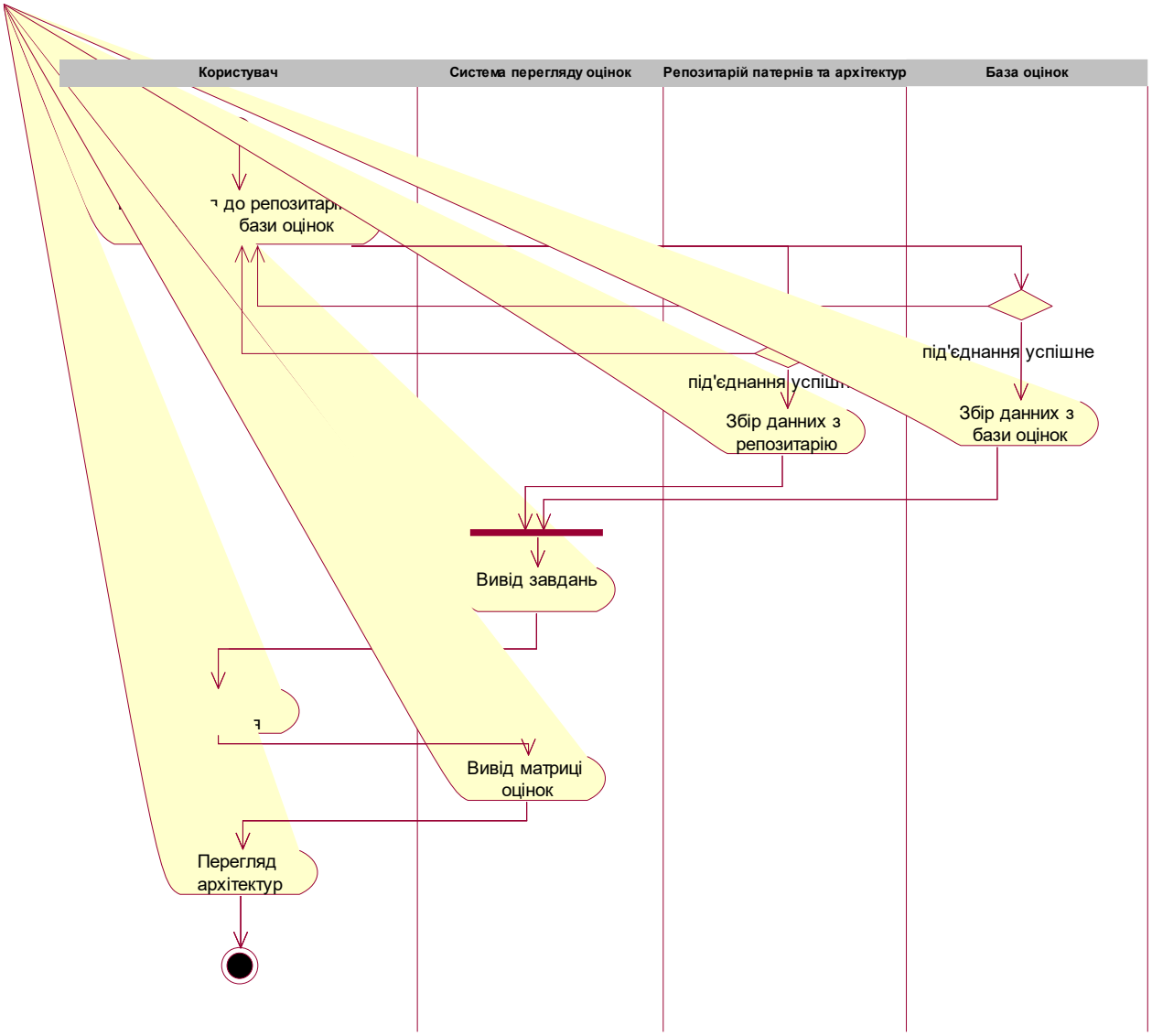


Рис. 2.13. Діаграма активності Проограми перегляду експертних оцінок

Дії (Activity):

1. Підключення до репозитарію шаблонів та бази оцінок –

2. Збір данних з репозитарію – збір данних з бази шаблонів та архітектур: завдань, побудованих альтернативних рішень…

3. Збір данних з бази оцінок – збір данних про проведені сесії кількісної оцінки

4. Вивід завдань – вивід списку сесій та завдань для вибору користувачем
5. Вибір завдання – вибір необхідної сесії
6. Вивід матриці оцінок – вивід матриці парних порівнянь альтернативних рішень
7. Перегляд архітектур – можливий перегляд архітектур

РОЗДІЛ 3

ПРАКТИЧНА РЕАЛІЗАЦІЯ СИСТЕМИ

3.1. Обгрунтування вибору інструментів розробки

IntelliJ – це одне з найпотужніших та найпопулярніших інтегрованих середовищ розробки (IDE) для Java. Воно створене та підтримується JetBrains та доступне у версії Community та Ultimate – у вигляді комерційної ліцензії. Ця багатофункціональна технологія IDE забезпечує швидку розробку та допомагає покращити якість коду.

Що таке IDE та його переваги?

Це комбінація декількох інструментів, які роблять процес розробки програмного забезпечення більш легким, надійним та менш схильним до помилок. Воно має такі переваги перед редактором простого тексту.

- Інтеграція з корисними інструментами, такими як компілятор, відладчик, система контролю версій, інструменти збирання, різні рамки, профілі додатків тощо.
- Підтримує навігацію кодом, доповнення коду, рефакторинг коду та генерування коду, що стимулює процес розробки.
- Підтримує тестування блоків, тестування інтеграції та покриття коду за допомогою плагінів.
- Забезпечує багатий набір плагінів для подальшого підвищення функціональності IDE.

IntelliJ IDEA має деякі найкращі продуктивні функції завершення коду Java. Його алгоритм прогнозування може точно припустити, що кодер намагається ввести, і доповнить його для нього, навіть якщо він не знає точного імені конкретного класу, члена чи будь-якого іншого ресурсу.

IntelliJ IDEA дійсно розуміє і глибоко розуміє ваш код, а також контекст кодера, що робить його таким унікальним серед інших ідентифікаторів Java IDE.

– Інтелектуальне завершення коду – підтримує контекстне завершення коду. Він дає список найбільш релевантних символів, застосовних у поточному контексті.

– Завершення ланцюгового коду – це вдосконалена функція заповнення коду, яка перераховує застосовні символи, доступні за допомогою методів або введення в поточному контексті.

– Статичне завершення учасника – дозволяє використовувати статичні методи або константи і автоматично додавати необхідні заяви про імпорт, щоб уникнути помилки компіляції.

– Виявлення дублікатів – він виявляє фрагменти коду, що повторюються, на ходу та дає користувачеві повідомлення / пропозиції про це.

– Огляди та швидкі виправлення – кожного разу, коли IntelliJ виявить, що ви збираєтеся помилитися, на цій же лінії вискакує невелике сповіщення про лампочку. Клацнувши по ньому, відображається список пропозицій.

Ергономіка розробника

IntelliJ IDEA розроблений на основі принципу кодування, що розробникам слід дозволяти писати коди з якомога меншим відволіканням. Ось чому в цьому випадку редактор – це єдине, що видно на екрані, з виділеними ярликами для всіх інших функцій, що не стосуються кодування.

– Середовище, орієнтоване на редактора – Швидкі спливаючі вікна допомагають перевірити додаткову інформацію, не виходячи з поточного контексту.

– Ярлики для всього – IntelliJ IDEA має комбінації клавіш для майже всього, включаючи швидкий вибір та перемикання між вікнами інструментів та багато іншого.

– Вбудований відладчик – Inline debugger дозволяє налагоджувати додаток у самому IDE. Це робить процес розробки та налагодження безпроблемним.

Вбудовані інструменти для розробників

Щоб допомогти розробникам в організації робочого процесу, IntelliJ IDEA пропонує їм дивовижний набір інструментів, який складається з декомпілятора, підтримки Docker, переглядача байт-кодів, FTP та багатьох інших інструментів розробника.

– Контроль версій – IntelliJ підтримує більшість популярних систем контролю версій, таких як Git, Subversion, Mercurial, CVS, Perforce та TFS.

– Інструменти побудови – IntelliJ підтримує Java та інші інструменти збирання, такі як Maven, Gradle, Ant, Gant, SBT, NPM, Webpack, Grunt та Gulp.

– Покриття бігу тесту та коду – IntelliJ IDEA дозволяє легко проводити тестування приладів. IDE включає тестові прогони та засоби покриття для основних тестових рамок, включаючи JUnit, TestNG, Spock, Cucumber, ScalaTest, spec2 та Karma.

– Декомпілятор – IntelliJ поставляється із вбудованим декомпілятором для класів Java. Коли ви хочете заглянути в бібліотеку, для якої у вас немає вихідного коду, ви можете це зробити без застосування сторонніх плагінів.

– Термінал – IntelliJ забезпечує вбудований термінал. Залежно від вашої платформи, ви можете працювати з командним рядком, наприклад PowerShell або Bash.

– Інструменти БД – IntelliJ надає інструменти для баз даних, які дозволяють вам підключатися до живих баз даних; виконувати запити; переглядати та оновлювати дані; і навіть керувати своїми схемами у візуальному інтерфейсі від самого IDE.

– Сервер додатків – IntelliJ підтримує основні сервери додатків: Tomcat, JBoss, WebSphere, WebLogic, Glassfish та багато інших. Ви можете розгорнути свої артефакти на серверах прикладних програм і налагодити розгорнуті програми в самому IDE.

– Підтримка Docker – Через окремий плагін IntelliJ пропонує спеціальне вікно інструментів, яке дозволяє підключатися до локальних машин Docker.

Компарація між Ultimate та Community Edition.

Ultimate Edition призначений для сприяння розвитку веб-сторінок та корпорацій, тоді як Community Edition розроблений для JVM та Android Development. Розглянемо кілька важливих моментів, які допоможуть зрозуміти відмінності двох редакцій продукту (таблиця 3.1)

Таблиця 3.1

Компарація версій IntelliJ

Особливість	Ultimate Edition	Community Edition
License	Комерційний	З відкритим кодом, Apache 2.0. для комерційного розвитку.
Java, Kotlin, Groovy, Scala	Підтримується	Підтримується
Android development	Підтримується	Підтримується
Maven, Gradle, SBT	Підтримується	Підтримується
Git, SVN, Mercurial, CVS	Підтримується	Підтримується
Detecting Duplicates	Підтримується	Не підтримується
Perforce, TFS	Підтримується	Не підтримується
JavaScript, TypeScript	Підтримується	Не підтримується
Java EE, Spring, GWT, Vaadin, Play, Grails, Other Frameworks	Підтримується	Не підтримується
Database Tools, SQL	Підтримується	Не підтримується

У всіх версіях IntelliJ IDEA має хороші функції що полегшують розробку. Наприклад, завершення коду Java. Його алгоритм прогнозування може точно припустити, що розробник намагається ввести, і доповнить його ввід автоматично, навіть якщо він не знає точного імені конкретного класу, члена чи будь-якого іншого ресурсу.

Інтерфейс середовища розробки IntelliJ IDEA показано на рисунку 3.1.

3.2. Вимоги до комп’ютера та операційної системи для застосування програмного комплексу

Програмний комплекс може бути використаний на персональному комп’ютері.

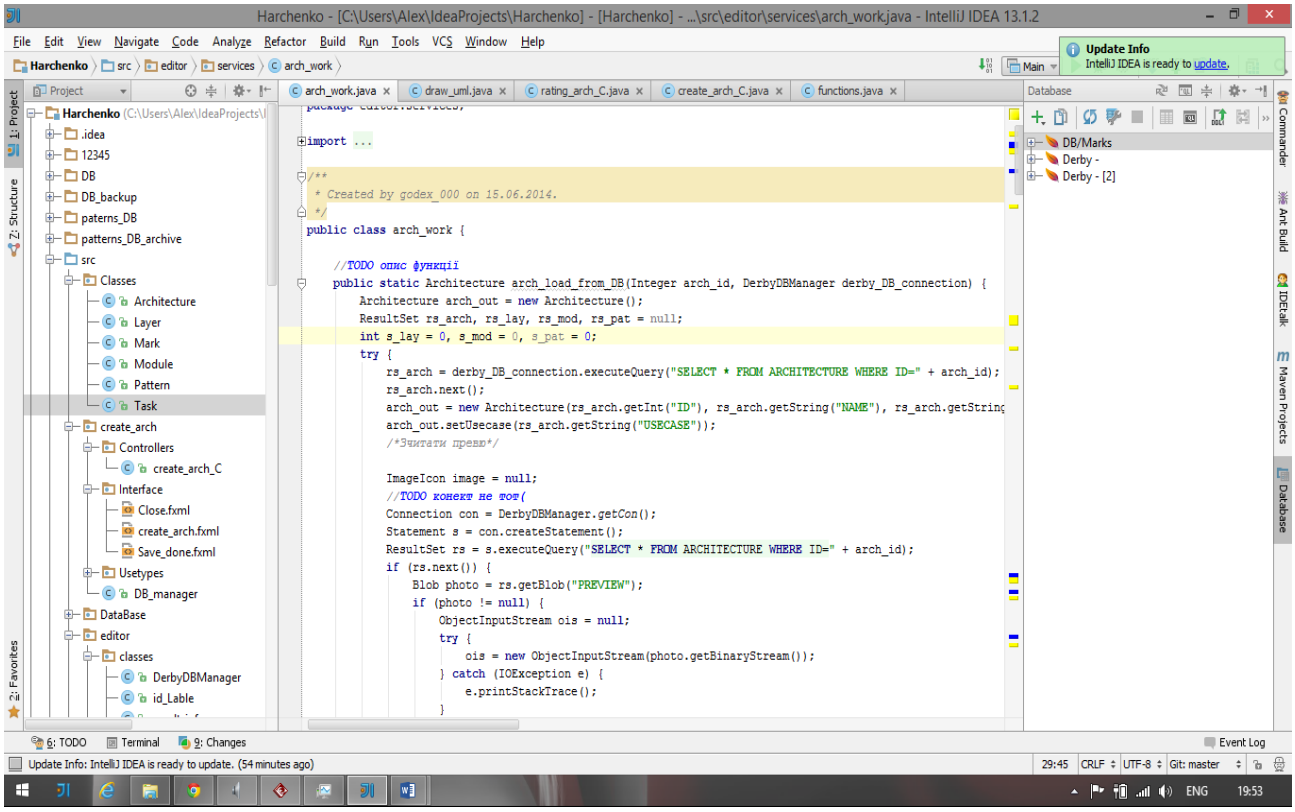


Рис. 3.1. Інтерфейс інтегрованого середовища розробки IntelliJ IDEA 13.1.1

Мінімальні характеристики:

– IBM сумісний комп’ютер класу Intel;

– Об’єм оперативної пам’яті (ОЗП) – 1 Гбайт;

– Об’єм доступного дискового простору – не менше 250 Мбайт;

– Операційна система – Microsoft Windows 8 і вище, Linux, Ubuntu, OS X;

– Наявність програмних засобів – програмне забезпечення графічної візуалізації graphviz v.2.38 і вище та Java Runtime Environment (JRE) 8 і вище.

Рекомендовані характеристики:

– Персональний IBM сумісний комп’ютер класу Intel Core i5 2xxx і вище;

– Об’єм оперативної пам’яті (ОЗП) – 4 Гбайт;

– Об’єм вільного дискового простору – 1 Гбайт;

– Операційна система – Microsoft Windows 8.1, 10;

– Наявність програмних засобів – програмне забезпечення графічної візуалізації graphviz v.2.38 і вище та Java Runtime Environment (JRE) 8 і вище.

Інсталяція програми

Для інсталяції програми в ОС Windows необхідно створити нову папку на жорсткому диску персонального комп’ютера та розпакувати в неї архів install.exe.

3.3. Методика роботи з системою

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

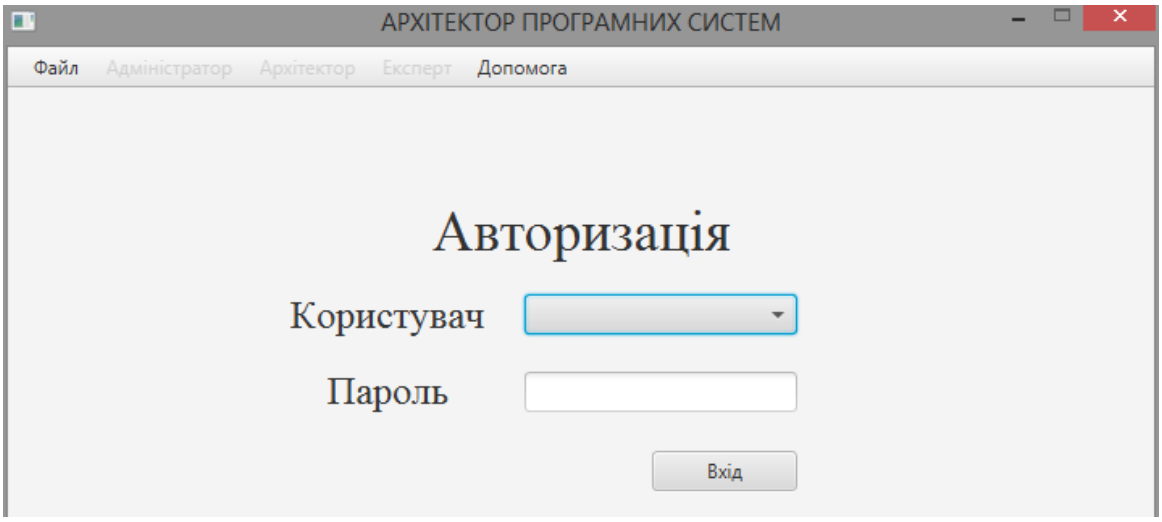


Рис. 3.2. Початкове вікно системи

Для початку роботи із системою користувач має обрати власну роль у системі і пройти етап авторизації.

3.4. Режим “Архітектор”

В разі вибору ролі Архітектора спочатку необхідно ввести пароль для нього. Після авторизації потрібно натиснути кнопку “Вхід” і вибрати операцію з головного меню для Архітектора, як показано на рисунку 3.4. Після цього почне працювати підсистема створення альтернативних програмних можливих пропозицій проектів архітектури проектованої системи .

Архітектор має функції “Створення альтернативних програмних можливих пропозицій проектів архітектури проектованої системи ” та “Кількісної оцінки архітектур”. Перша операція слугує для створення нових множин альтернативних програмних можливих пропозицій проектів архітектури проектованої системи них проектів програмних засобів і забезпечується роботою підсистеми створення альтернативних програмних можливих пропозицій проектів архітектури проектованої системи (п. 2.2.). Друга операція необхідна для контролю та оцінки вже сформованих наборів альтернативних програмних можливих пропозицій проектів архітектури

проектованої системи . Ця функція забезпечується підсистемою кількісної оцінки альтернативних проектів архітектури проектованої системи (Експерт), яка буде розглянута у п.3.3.2.

Вибір функції “Створення альтернативних програмних можливих пропозицій проектів архітектури проектованої системи ” показаний на рисунку 3.3. В разі вибору цієї операції з’являється головне вікно підсистеми створення альтернативних програмних можливих пропозицій проектів архітектури проектованої системи , яке показано на рисунку 3.4. Перед початком роботи в підсистемі, Архітектору треба під’єднатися до БД, в якій зберігається репозиторій шаблонів відповідних архітектур, що показано на рисунку 3.5.

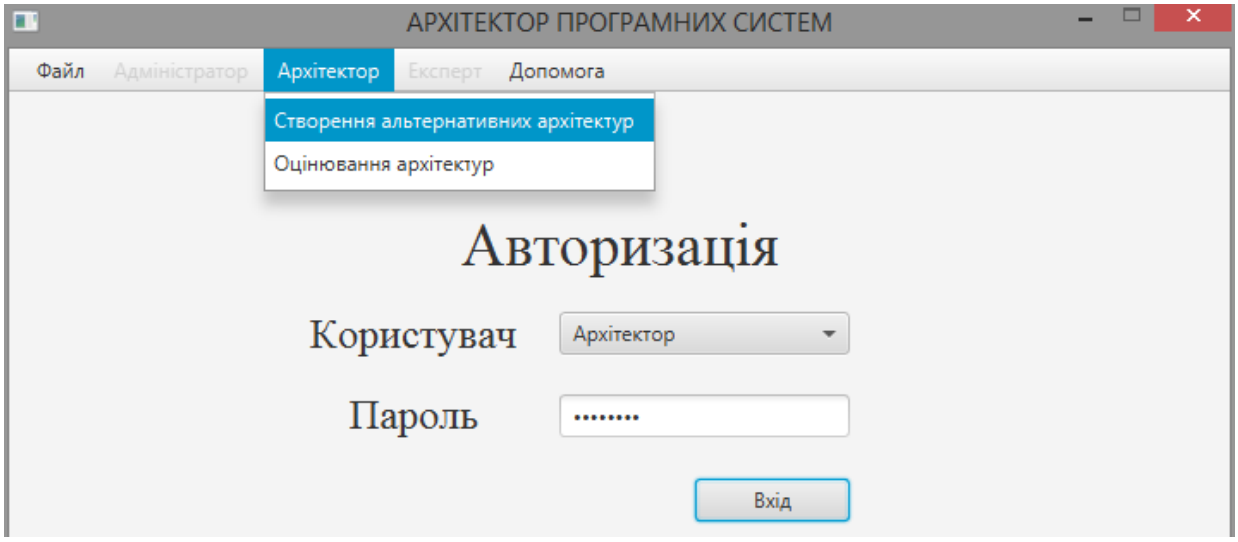


Рис. 3.3. Авторизація у системі та вибір ролі Архітектора

Після вибору пункту головного меню підсистеми: Файл і Підключення до БД, з’явиться вікно вибору БД, що зображене на рисунку 3.6.

Після підключення до БД будуть завантажені типи варіантів архітектурних проектів програмних засобів, які зберігаються в БД. По завершенню вибору варіанту архітектури, натискаємо обрати.

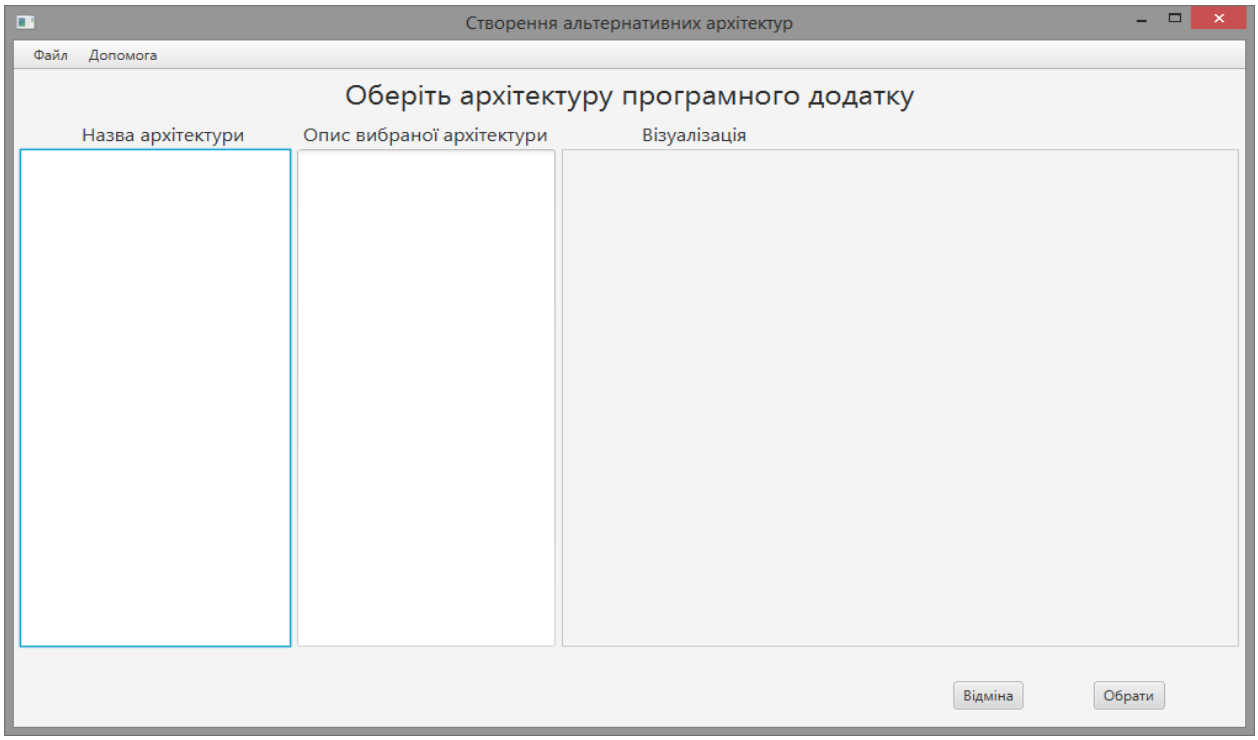


Рис. 3.4.Підсистема створення альтернативних рішень програмних архітектур

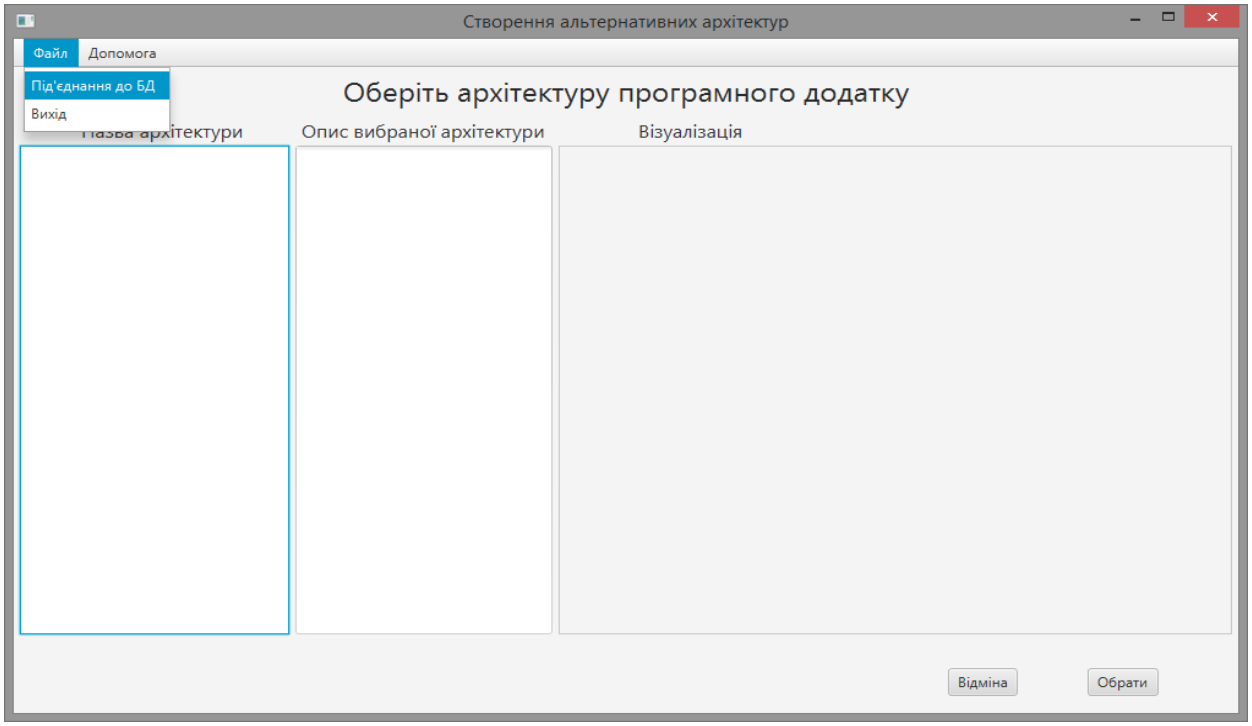


Рис. 3.5. Підключення до БД при створенні програмних архітектур

При виборі довільної архітектури із поданого списку візуалізується текстовий опис та схематичне зображення її шарів і модулів (візуалізація архітектури), як показано на рисунку 3.7.

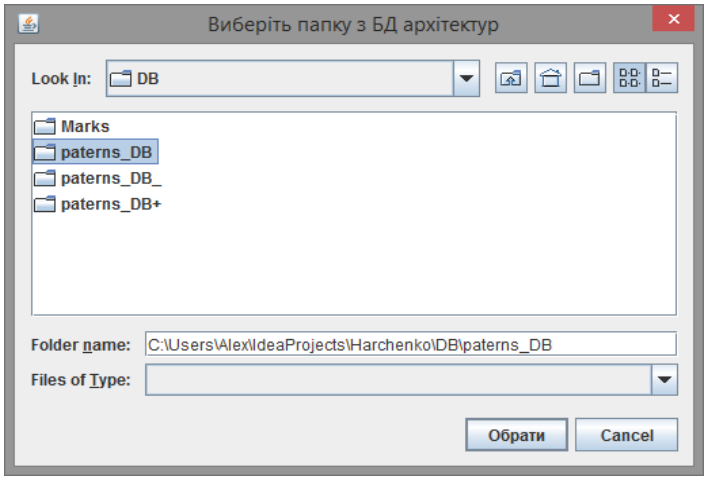


Рис. 3.6. Вікно вибору архітектури з БД

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

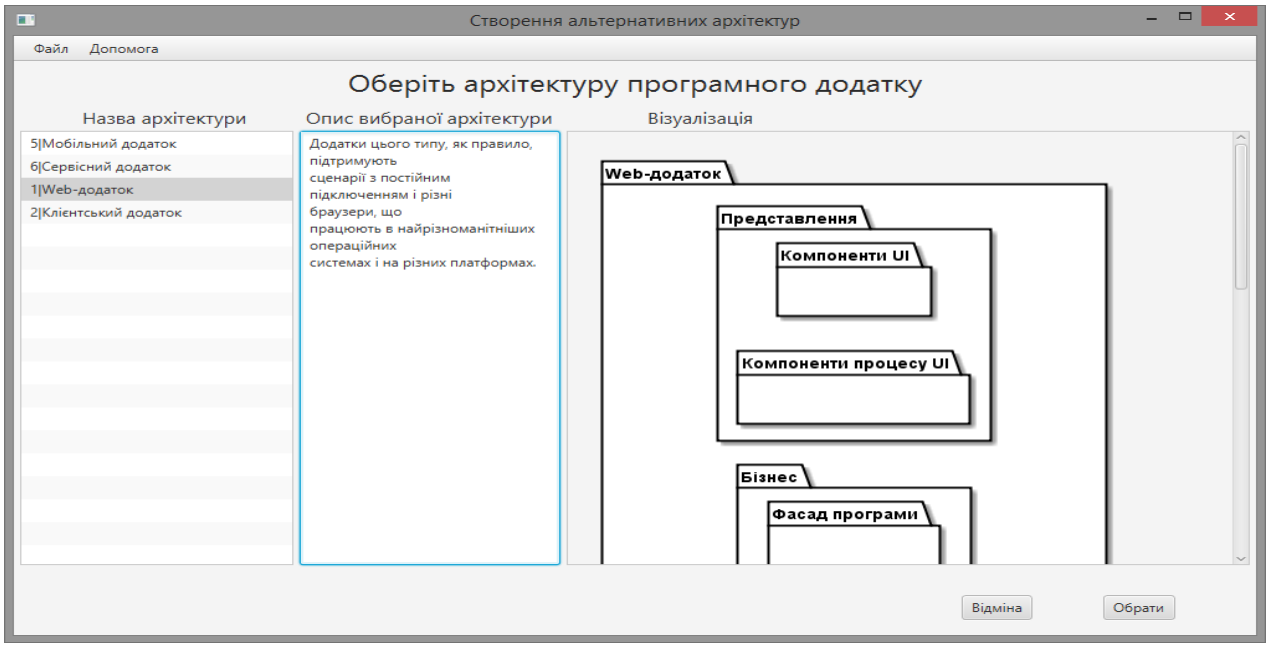


Рис. 3.7. Список архітектур для вибору

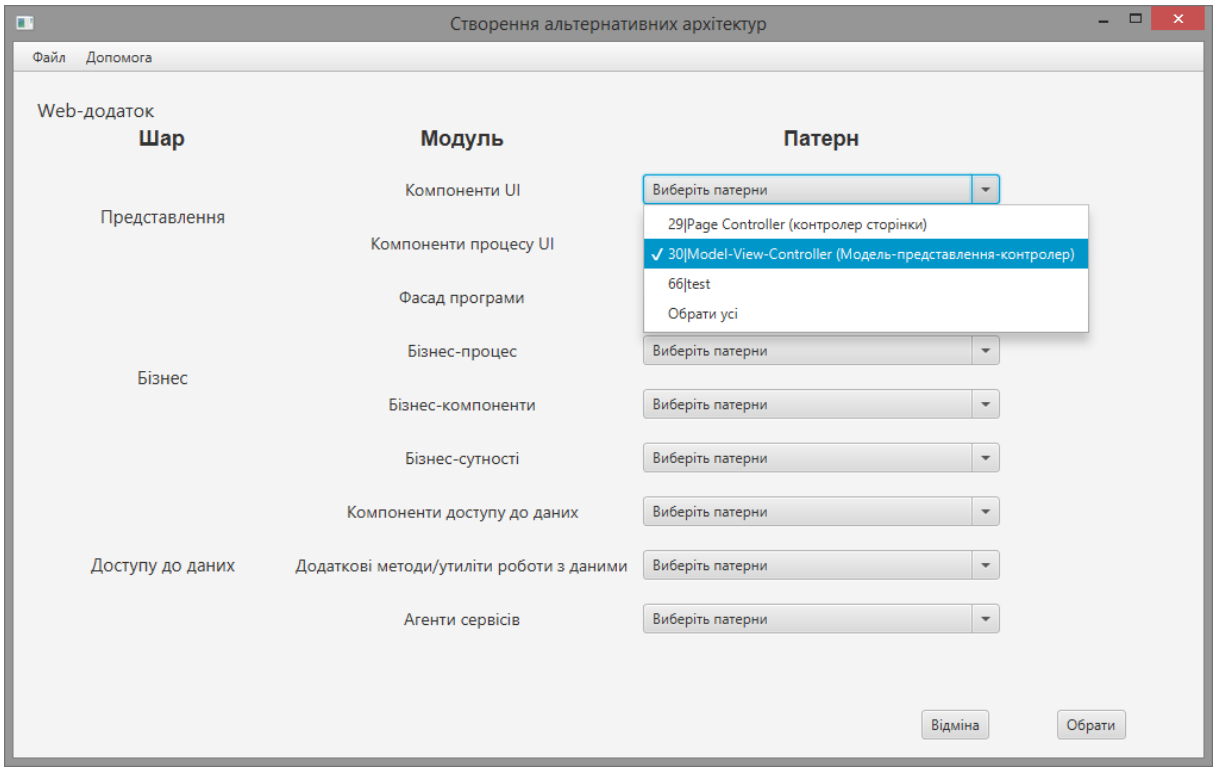


Рис 3.8. Вибіршаблонів для архітектури ПЗ

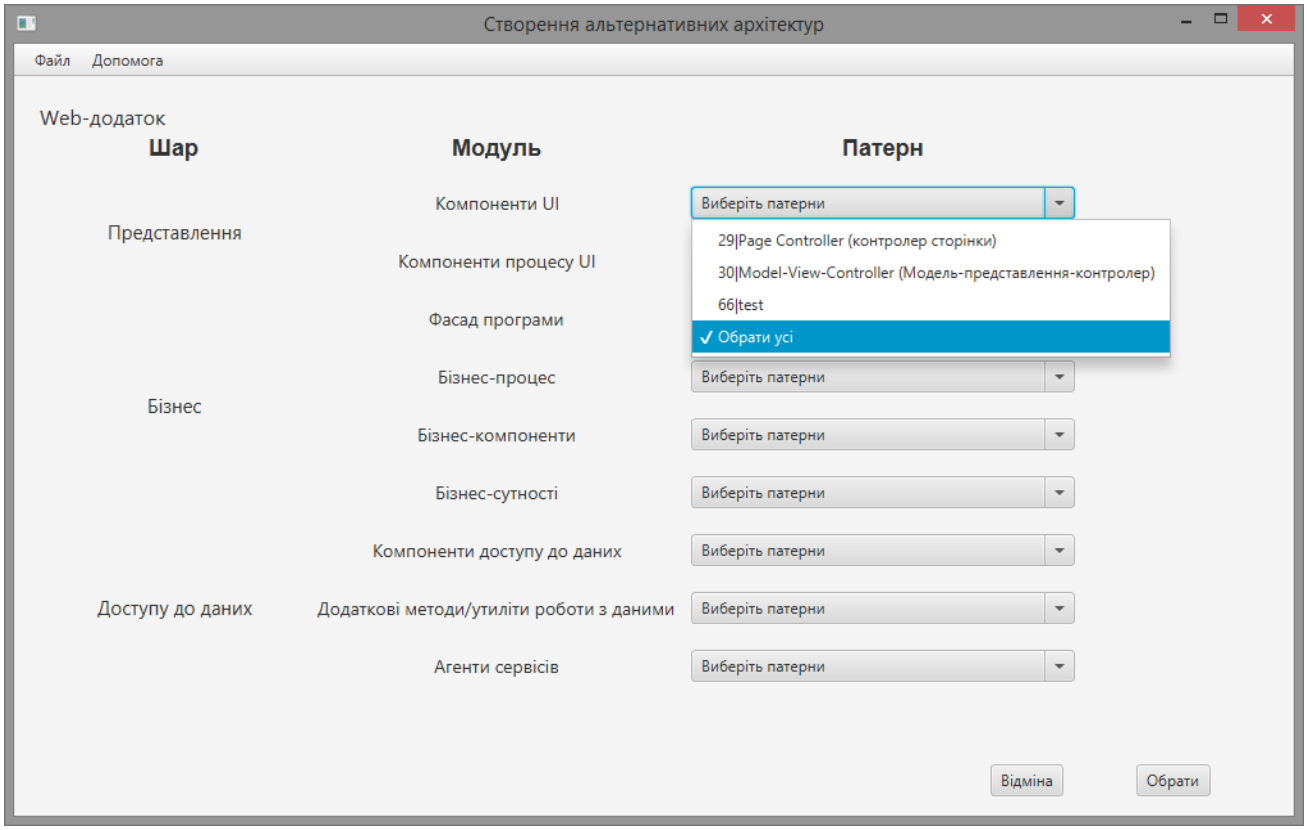


Рис. 3.9. Мультивибір варіантів шаблонів

По завершенню вибору шаблонів демонструється результат вибору, що показано на рисунку 3.10 і рисунку 3.11.

.

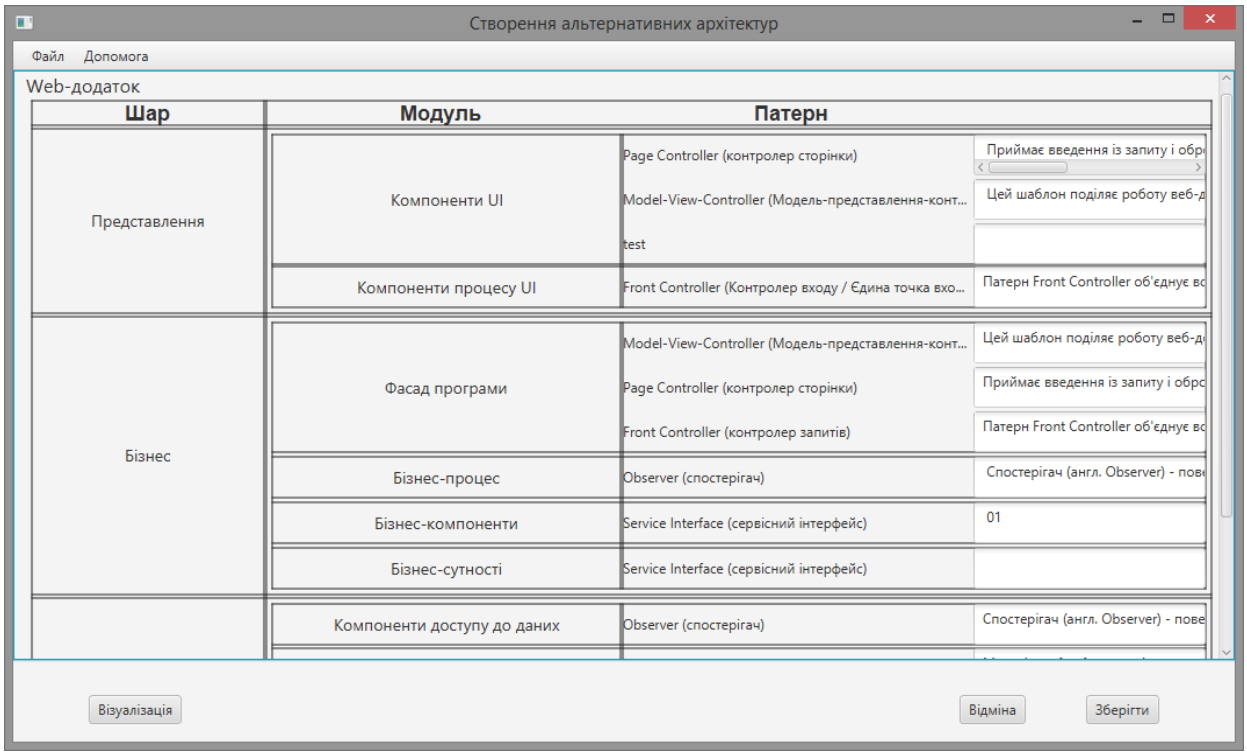


Рис. 3.10. Вікно обраних шаблонів

Після цього можна зберегти множину варіантів архітектур у базу даних, для подальшого їх кількісної оцінки Експертами. Для цього вводимо назву задачі для оцінки та її опис і зберігаємо, що показано на рисунку 3.12. По закінченню збереження видається вікно, яке показано на рисунку 3.13.

3.5. Режим “Експерт”

У разі вибору ролі Експерта необхідно ввести пароль. Після авторизації натискаємо кнопку “Вхід” і обираємо операцію з головного меню Експерта, що показано на рисунку 3.14. Після цього почне працювати «Підсистема кількісної оцінки альтернативних проектів архітектури проектованої системи».

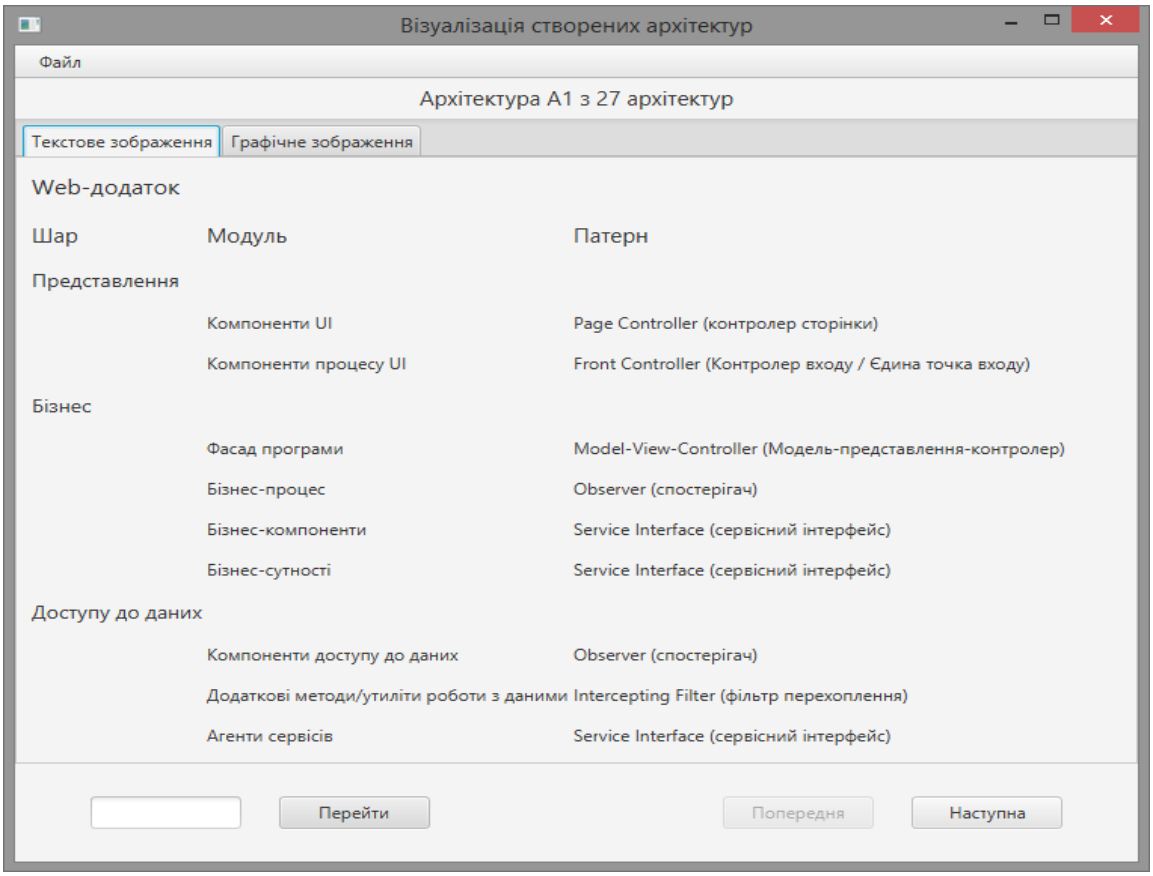


Рис. 3.11. Вікно візуалізації обраних шаблонів

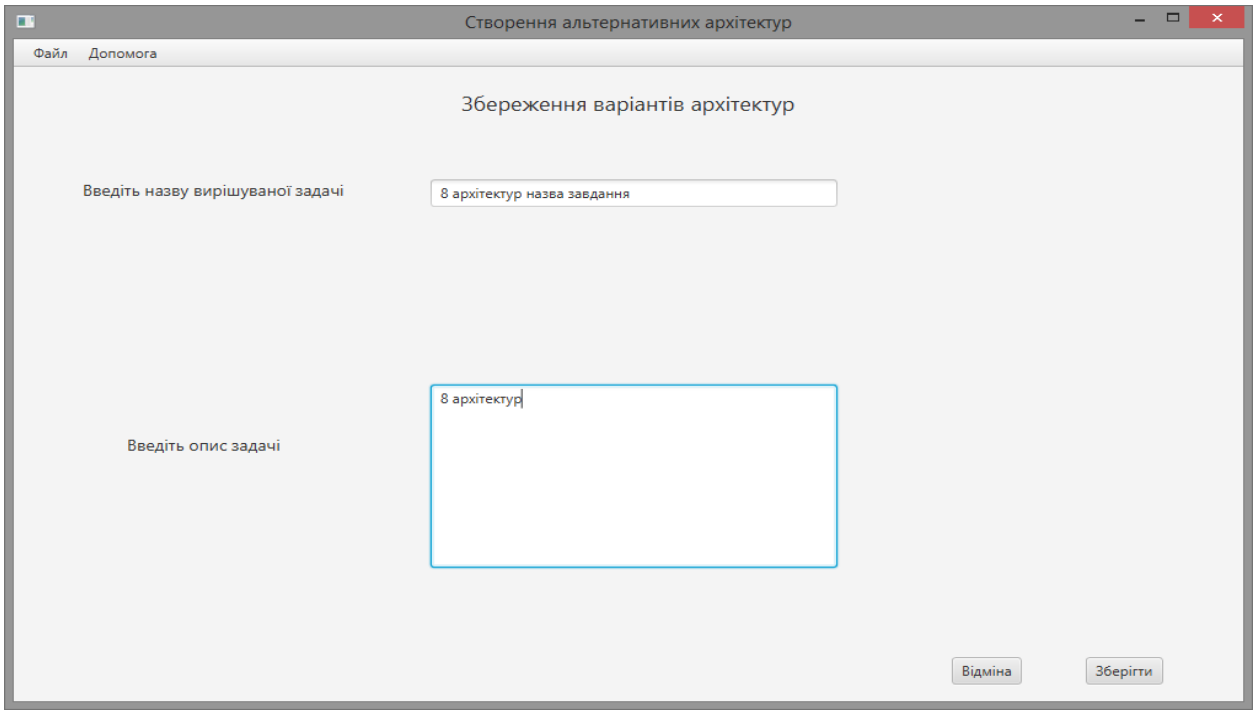


Рис. 3.12. Збереження задачі з обраними архітектурами

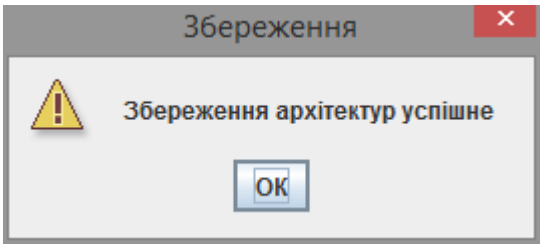


Рис. 3.13. Індикація для успішного збереження задачі

Експерт має лише одну функцію – “Кількісної оцінки альтернативних проектів архітектури проектованої системи”. Ця операція слугує для кількісної оцінки вже сформованої множини варіантів альтернативних програмних можливих пропозицій проектів архітектури проектованої системи програмного додатка і забезпечується роботою підсистеми кількісної оцінки альтернативних проектів архітектури проектованої системи (п.2.3.).

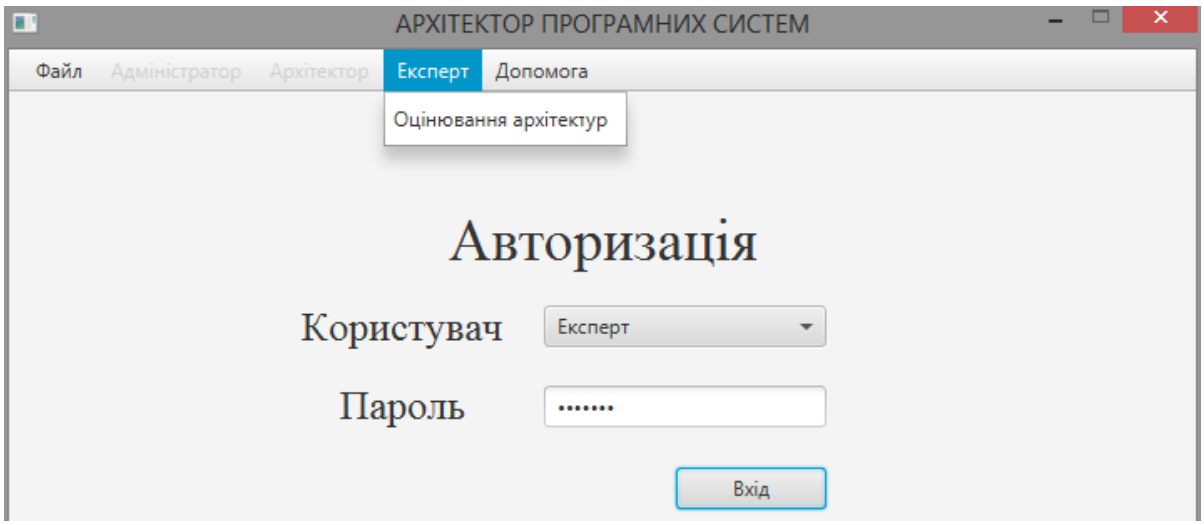


Рис. 3.14. Авторизація у системі

Якщо потрібно закінчити роботу системи, слід у головному меню вибрати “Файл” та “Вихід”, як показано на рисунку 3.15.

Вибір функції “Кількісної оцінки альтернативних проектів архітектури проектованої системи” показано на рисунку 3.16. У разі вибору цієї операції з’являється головне вікно підсистеми кількісної оцінки альтернативних проектів архітектури проектованої системи, яке показано на рисунку 3.17.

Перед початком роботи Експерту треба під’єднатися до БД, в якій зберігається репозиторій шаблонів архітектур. Це виконується шляхом вибору пунктів головного меню “Файл” і “Підключення до БД”, що показано на рисунку 3.17.

Після підключення до БД будуть завантажені типи варіантів архітектурних проектів програмних засобів. Далі потрібно вибрати завдання (задачу) для оцінки заздалегідь підготовлених Архітектором варіантів альтернативних програмних можливих пропозицій проектів архітектури проектованої системи , що показано на рисунку 3.18. При виборі відповідної назви буде виведений опис задачі, як показано на рисунку 3.19. Після цього обираємо критерій для кількісної оцінки варіантів альтернативних програмних можливих пропозицій проектів архітектури проектованої системи , що показано на рисунку 3.20.

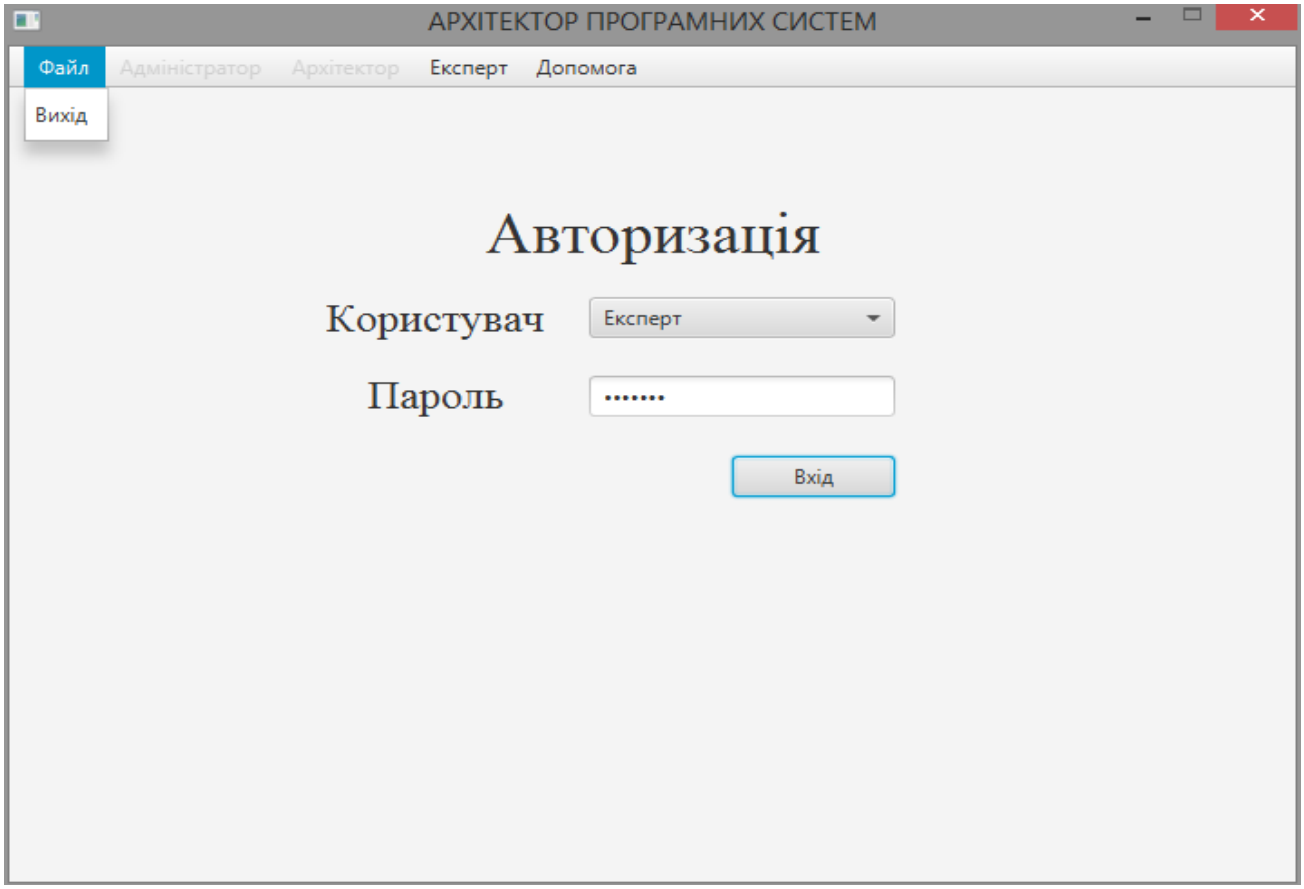


Рис. 3.16. Завершення роботи

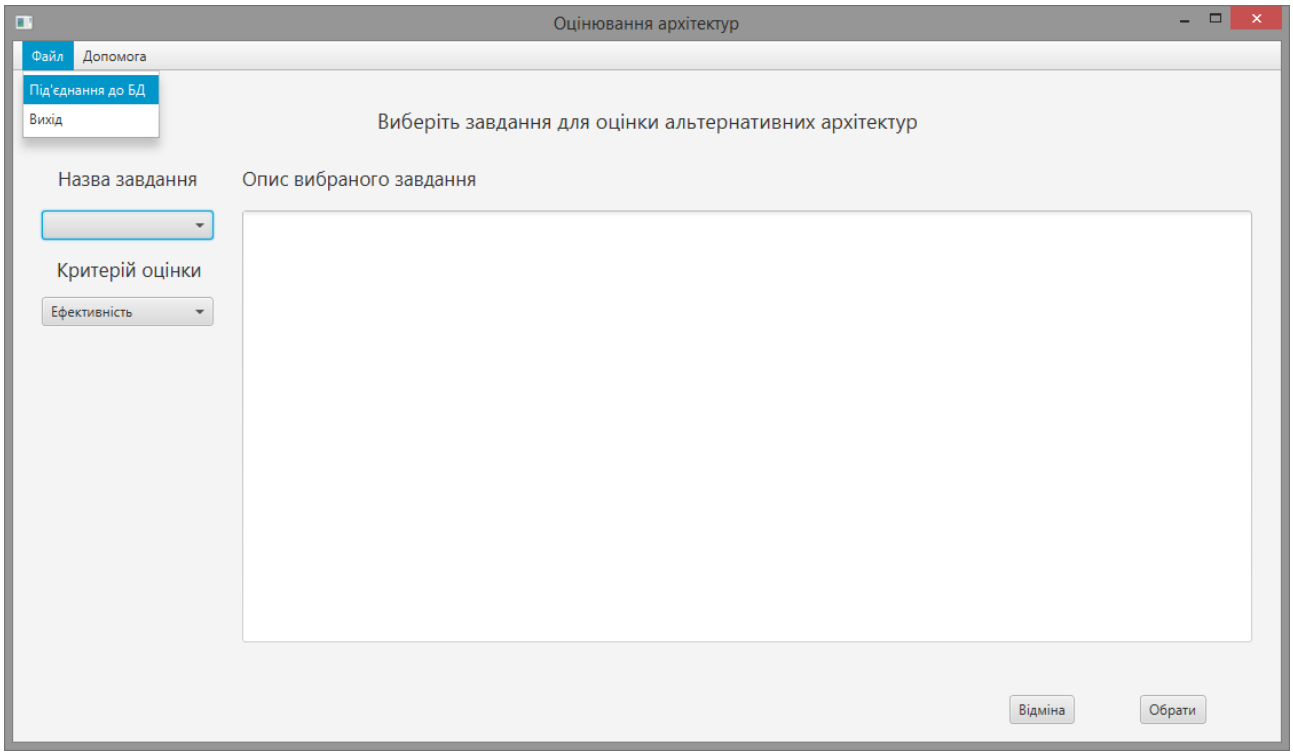


Рис. 3.17. Підключення ролі "Експерт" до БД

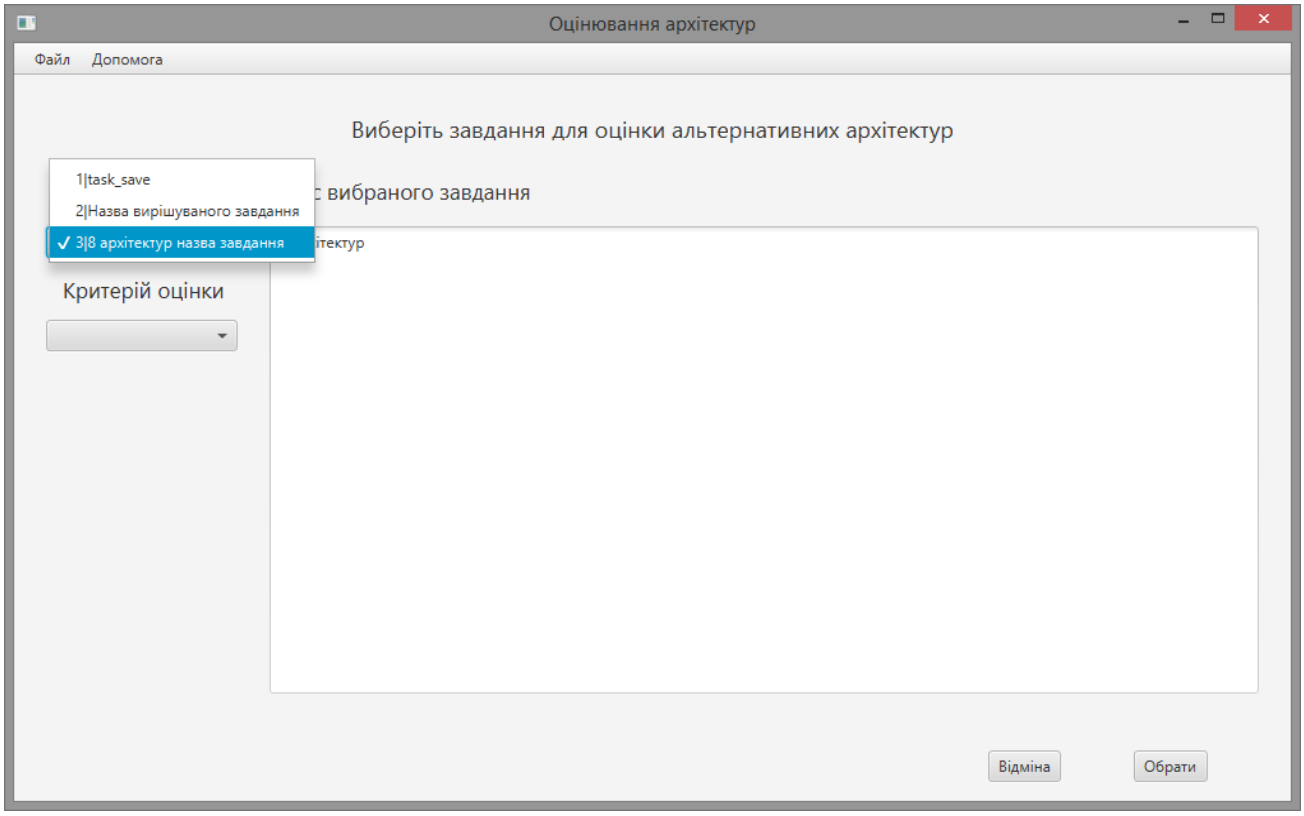


Рис. 3.18. Завдання на кількісної оцінки альтернативних програмних можливих пропозицій проектів архітектури проектованої системи

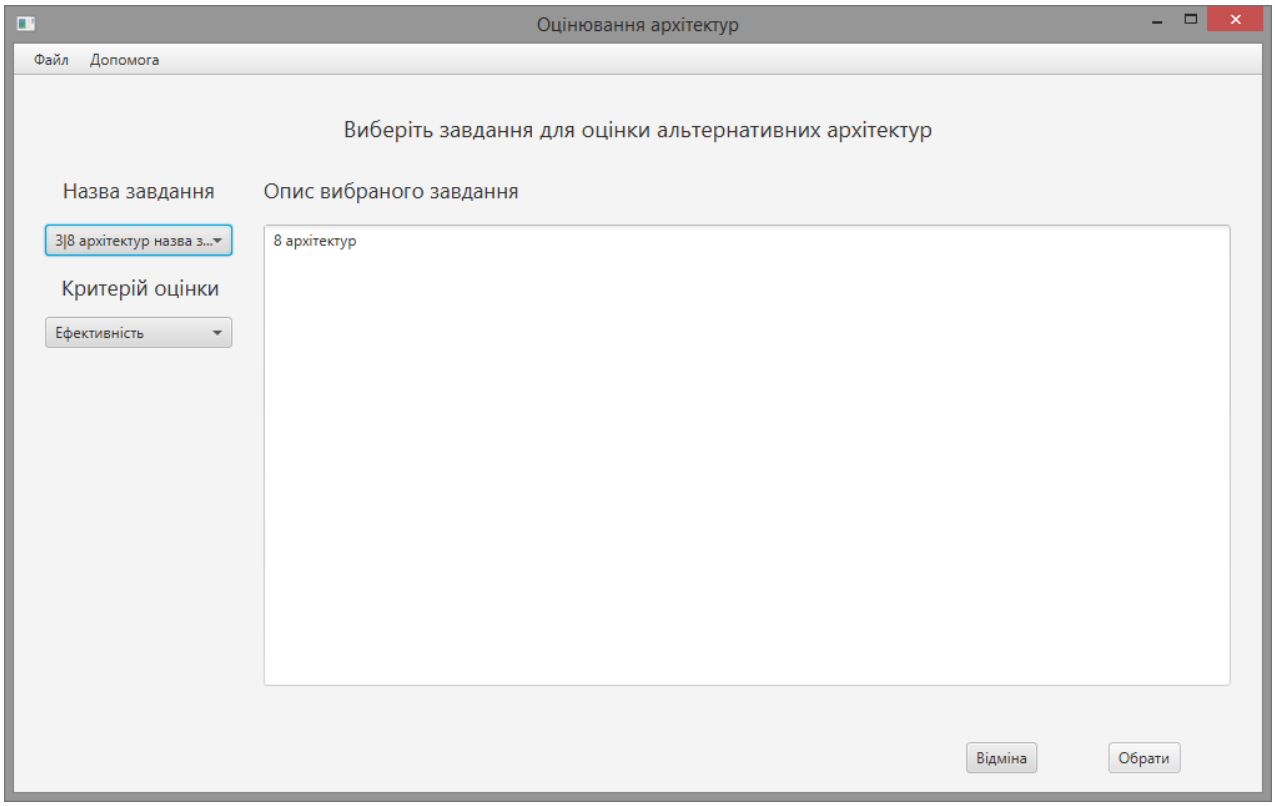


Рис. 3.19. Задача кількісної оцінки прогарнмих альтернативних архітектур

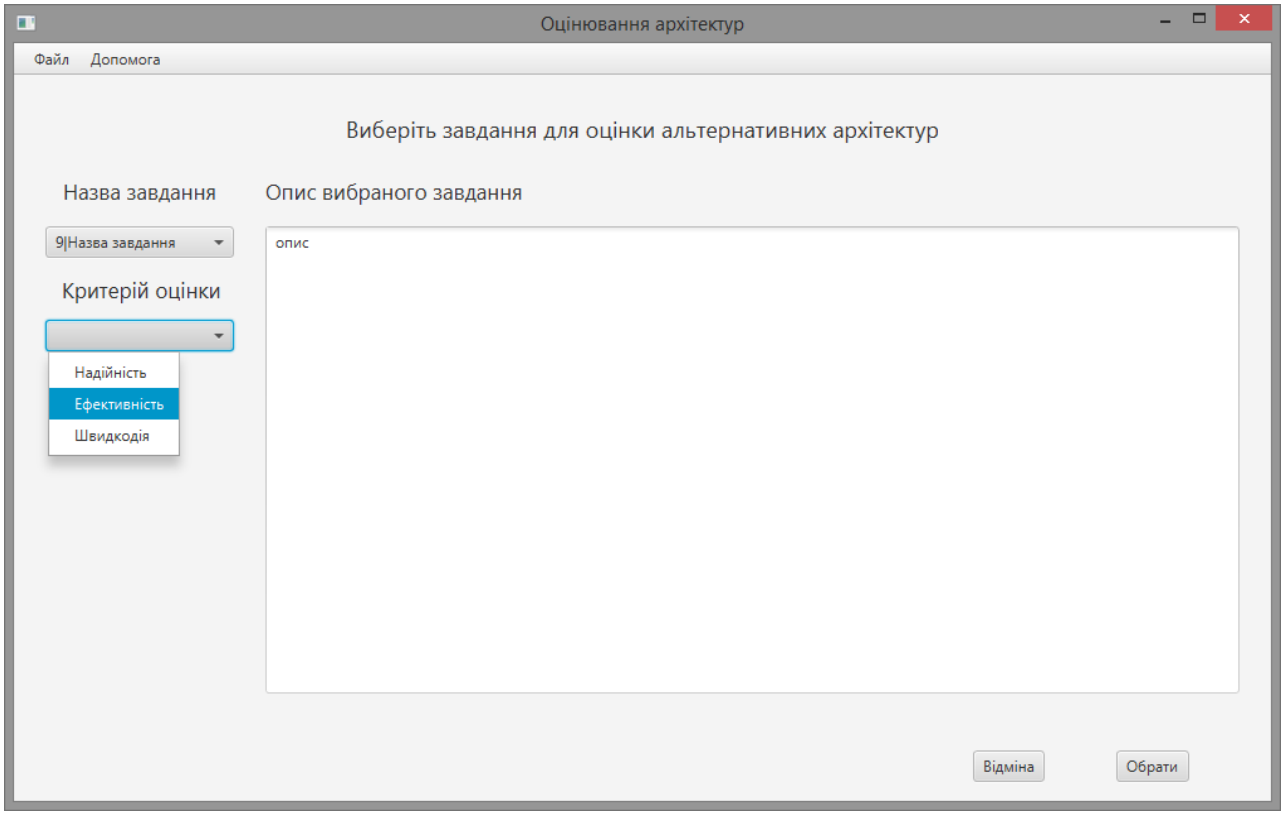


Рис. 3.20. Критерій якості для кількісної оцінки архітектур

Після вибору критерію будуть виводитися по парно одни з одним різні варіанти альтернативних програмних можливих пропозицій проектів архітектури проектованої системи з можливістю їх попарного порівняння у шкалі від 1 до 9 по аналогії до МАІ. Архітектури представляються як у текстовому табличному вигляді (назва типу програмного додатка, назва шару, назва модуля, назви шаблонів), так і у графічному вигляді (діаграми класів шаблонів, модулів і шарів додатку). Текстовий та графічний вигляд вікна кількісної оцінки альтернативних програмних можливих пропозицій проектів архітектури проектованої системи них рішень програмного продукту показано на рисунку 3.21 і рисунку 3.22.

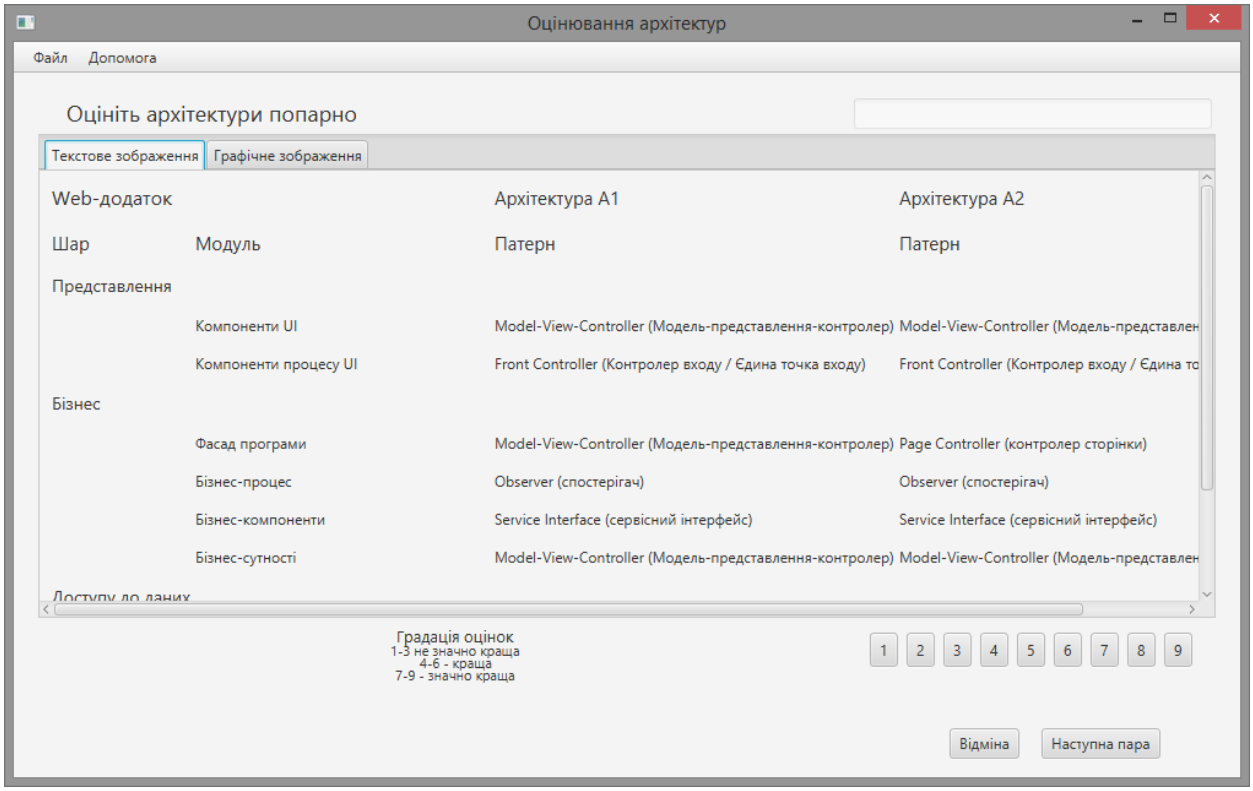


Рис. 3.21. Кількісної оцінки альтернативних програмних можливих пропозицій проектів архітектури проектованої системи них рішень програмного продукту в текстовому текстовий вигляді

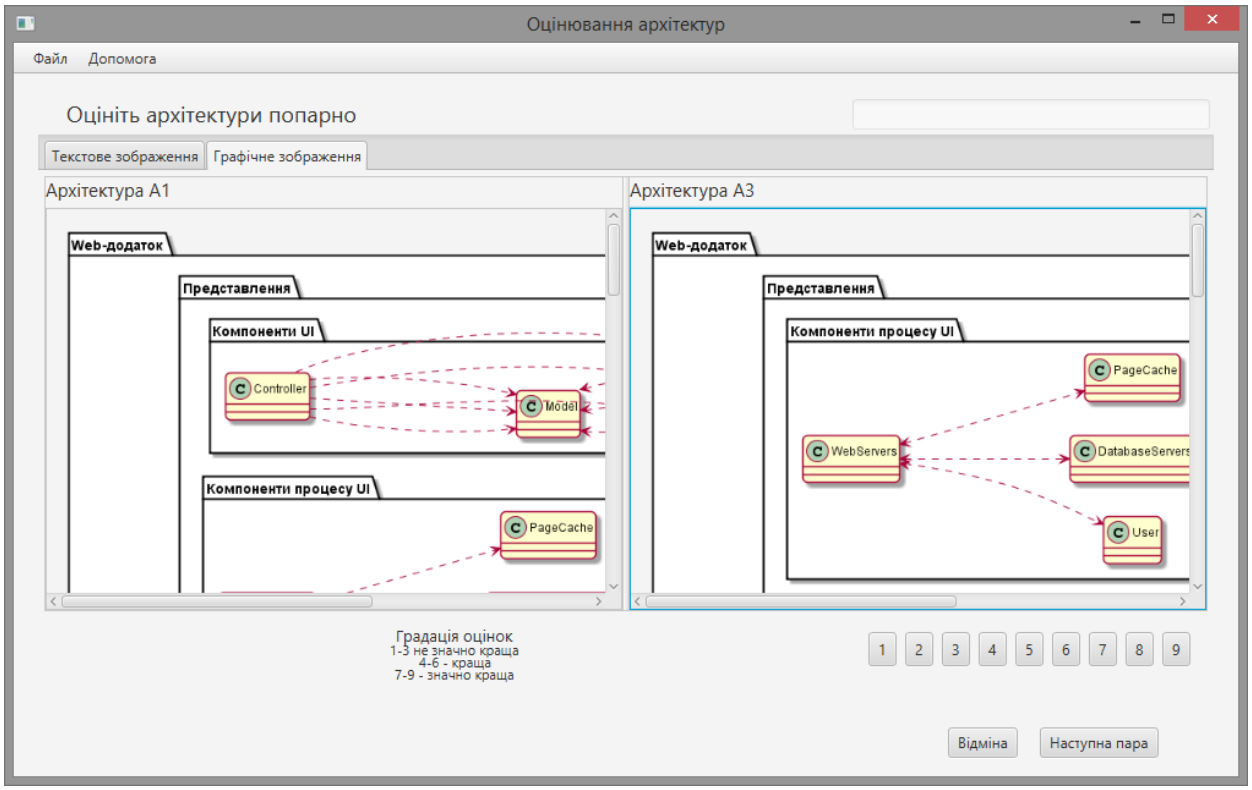


Рис. 3.22. Вікно кількісної оцінки альтернативних програмних можливих пропозицій проектів архітектури проектованої системи них рішень програмного продукту додатку (графічний вигляд)

Для виставлення оцінки Експерт натискає на кнопку відповідної оцінки, а при неможливості оцінити – натискає кнопку “Наступна пара”. По закінченню кількісної оцінки буде видане інформативне вікно, яке показане на рисунку 3.23.

Після закінчення кількісної оцінки виставлені оцінки поточної сесії будуть збережені у відповідних таблицях БД, а Експерт буде переведений у початкове головне вікно системи для подальшого вибору користувача, або виходу із системи.

3.6. Режим перегляду виставлених експертних оцінок

В режимі перегляду оцінок можливо переглянути виставлені експертами оцінки та архітектури яким ці оцінки були виставлені (див. рис. 3.24.). Після запуску програми проводиться операція підключення до

репозитарію шаблонів та архітектур, після чого підключається база виставлених експертних оцінок «Marks», яка зазвичай знаходиться в директорії DB програмного комплексу (див. рис. 3.25.).

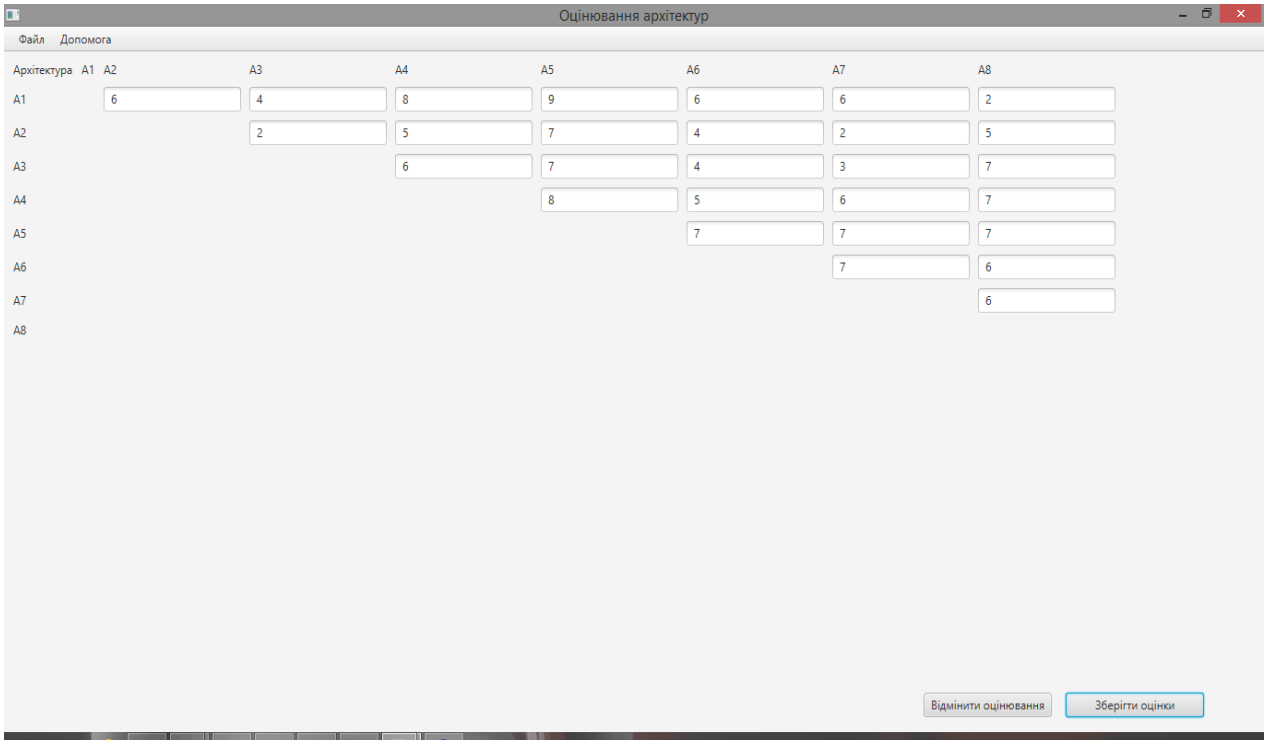


Рис. 3.23. Матриця порівняльних оцінок множини альтернативних програмних можливих пропозицій проектів архітектури проектованої системи додатку

Якщо бази валідні, можна буде вибрати сесію кількісної оцінки експерта, переглянути нотатки експерта, опис вирішуваної задачі та критерій по якому проводилося кількісної оцінки (див. рис. 3.26). Вибравши сесію переходимо до перегляду безпосередніх неопрацованих попарних оцінок альтернативних програмних можливих пропозицій проектів архітектури проектованої системи (правити оцінки неможливо). При необхідновті можливо переглянути архітектури натиснувши на кнопку цікавлючої архітектури (див. рис. 3.27).

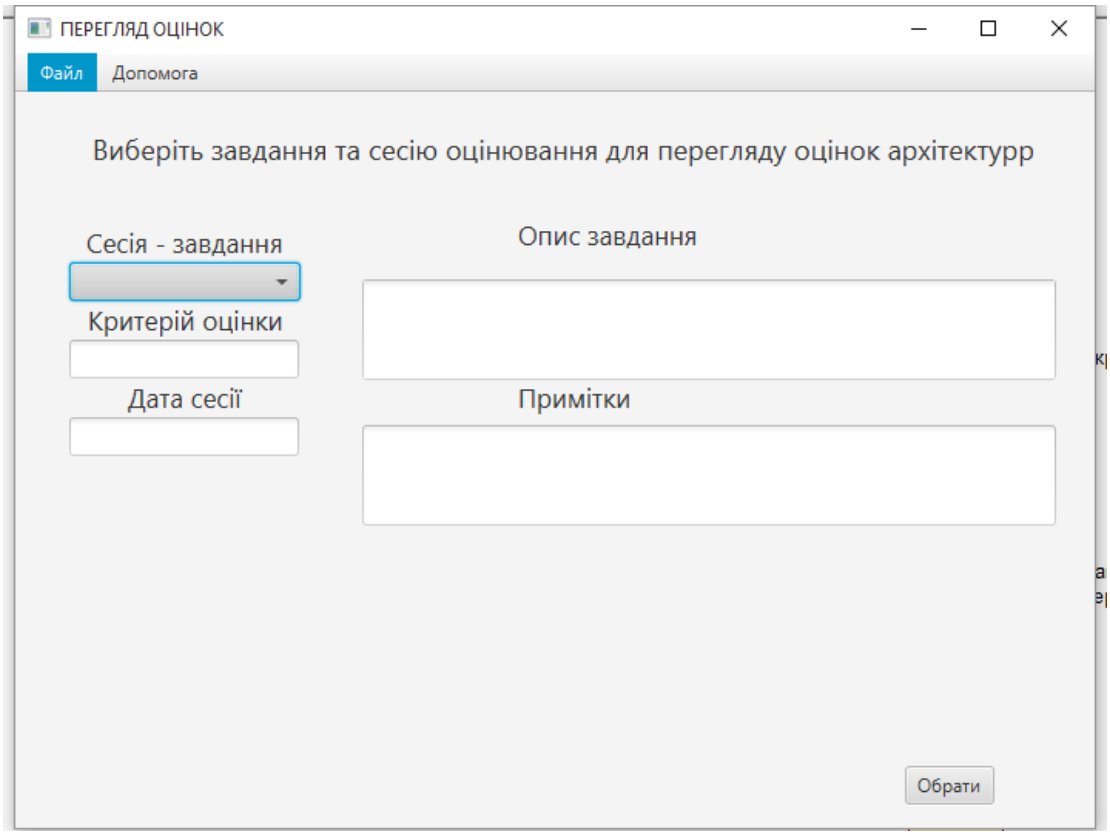


Рис. 3.24. Стартова сторінка режиму перегляду оцінок

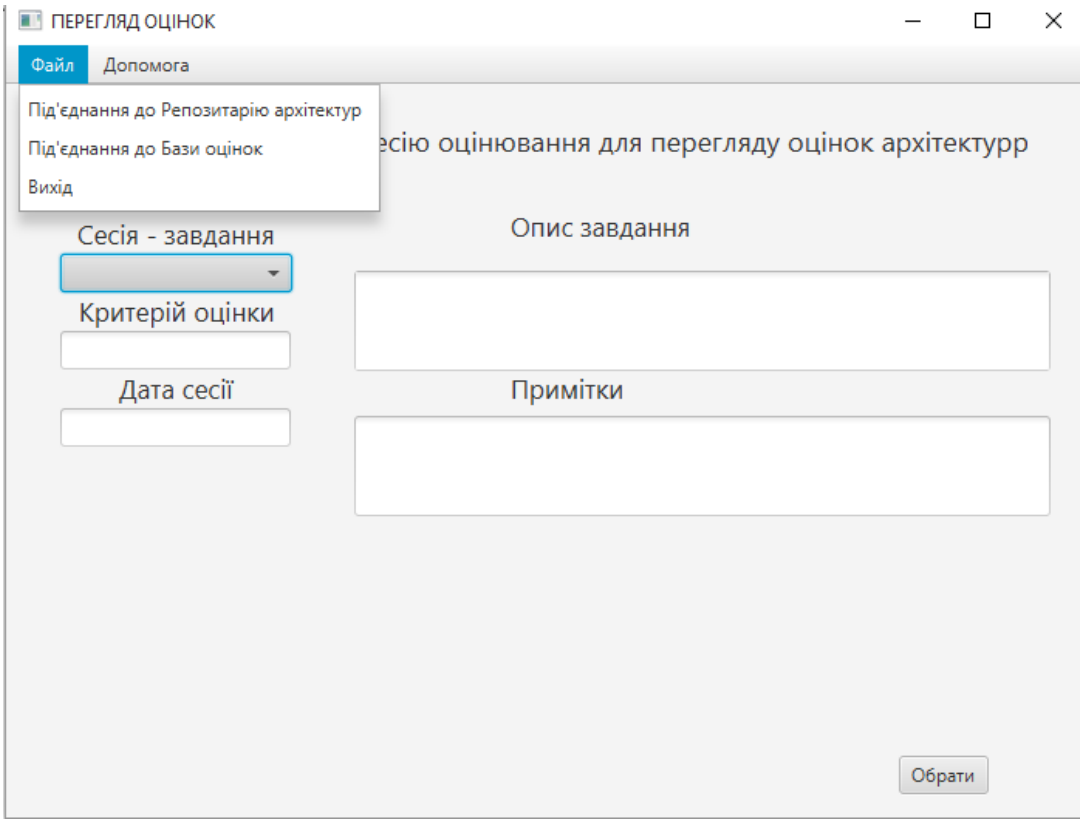


Рис. 3.25. Інтерфейс вибору репозитарію шаблонів та бази оцінок

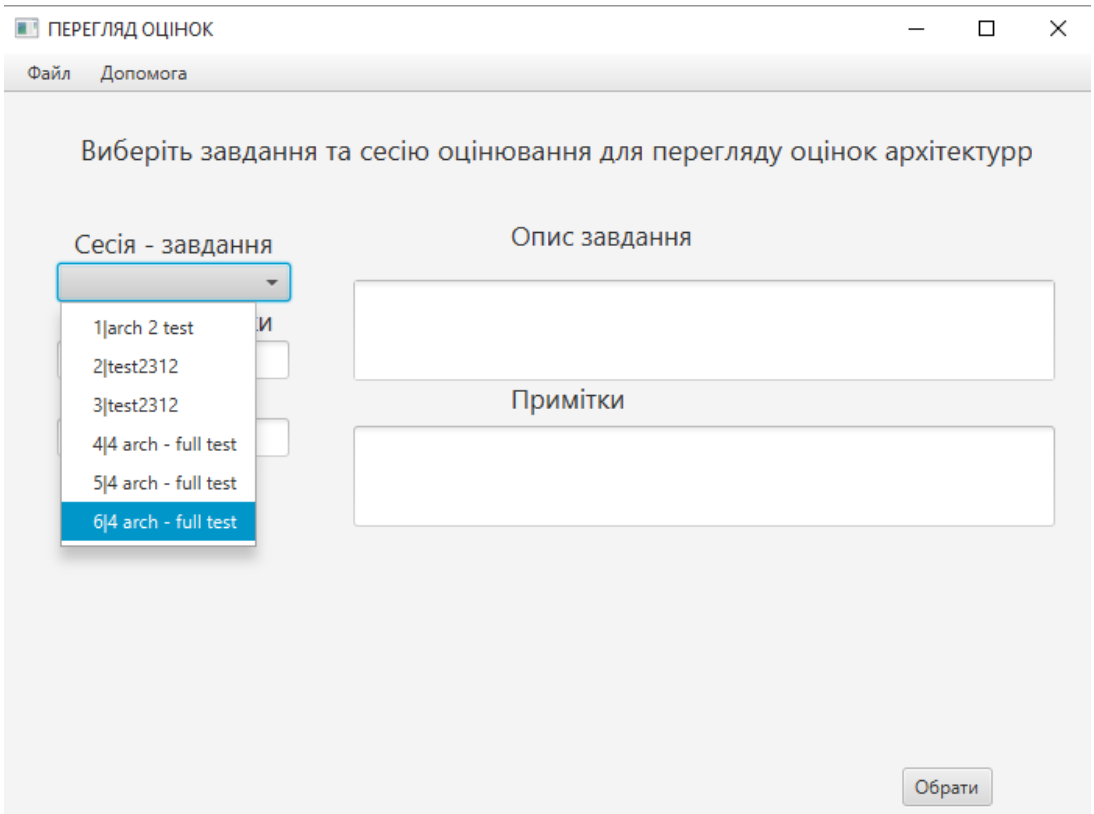


Рис. 3.26. Вибір сесії кількісної оцінки (SessionID|назва вирішунованої задачі)

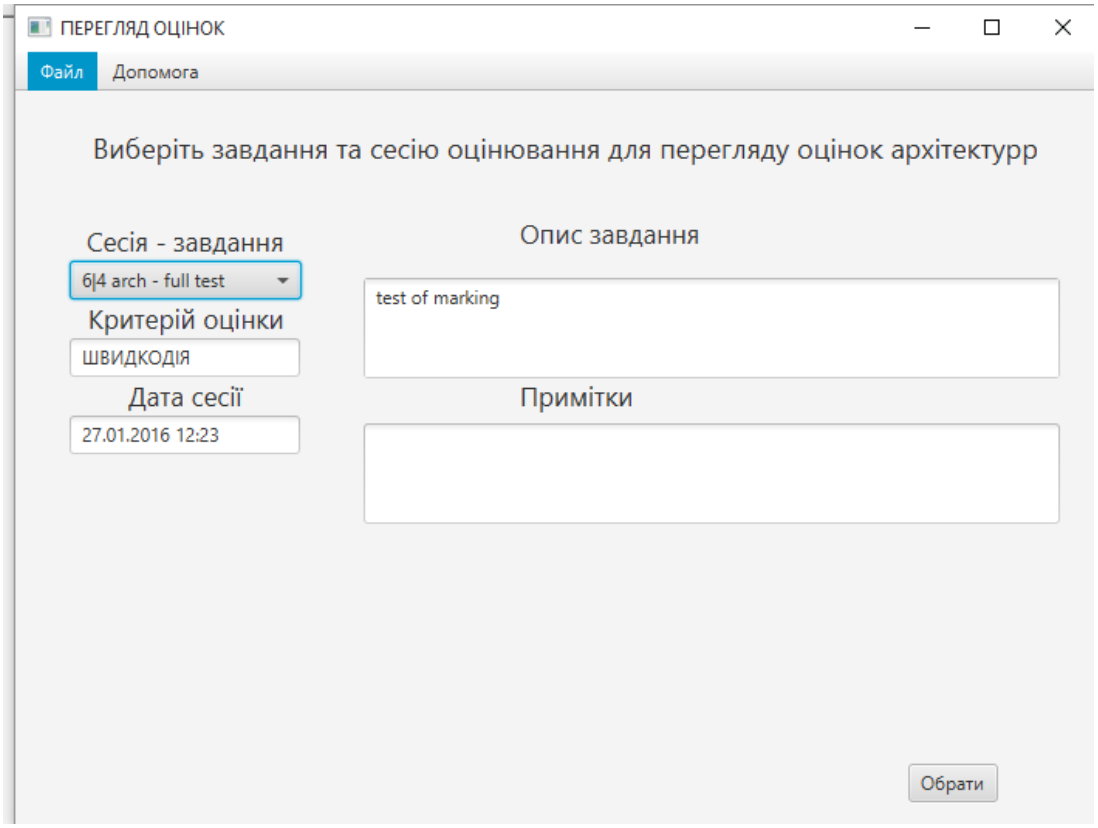


Рис. 3.27. Вікно з вибраною сесією

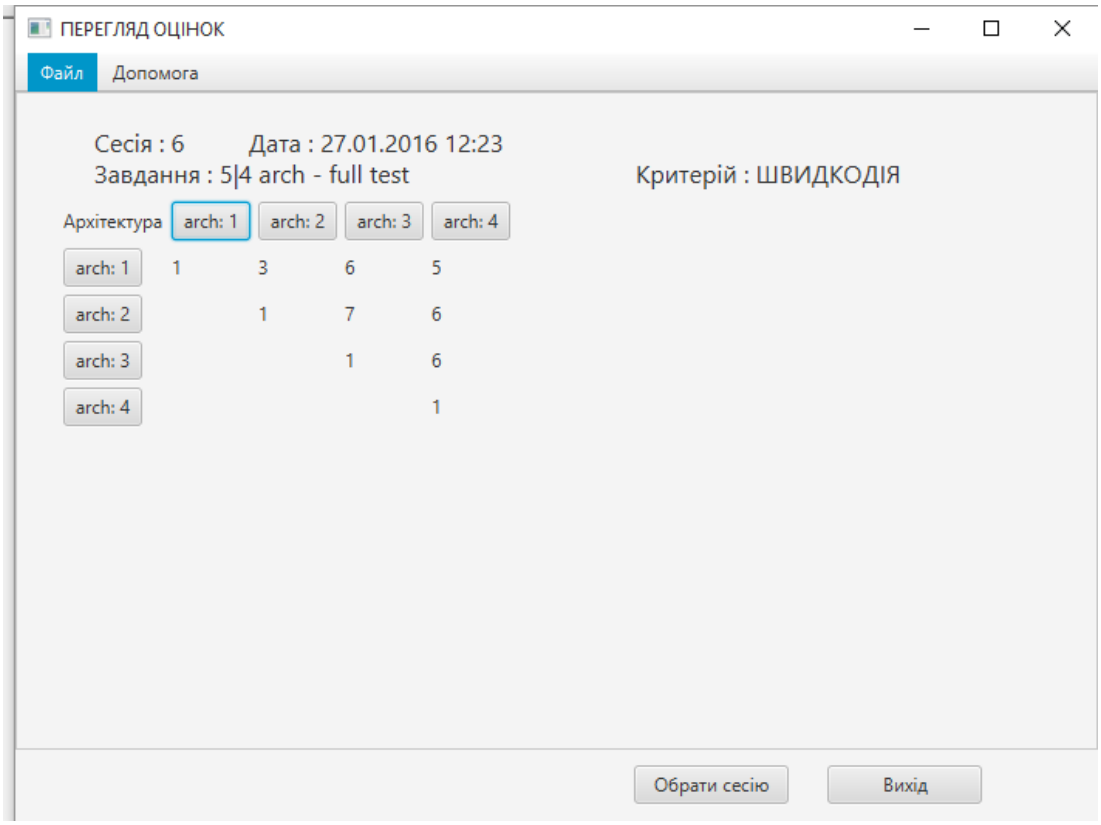


Рис. 3.28. Вивід матриці попарних порівнянь (перегляд оцінок)

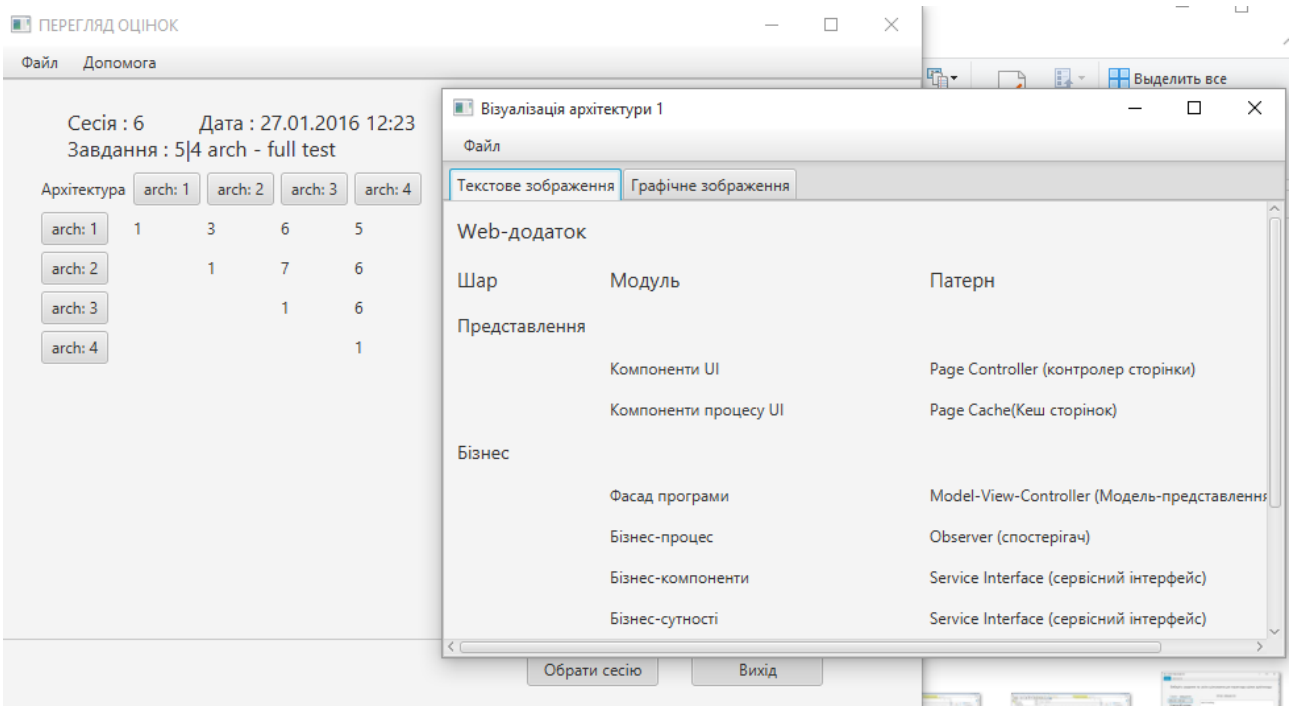


Рис. 3.29.Перегляд архітектури в текстовому вигляді

(можливий перегляд у графічному)

3.7. Режим виставлення критеріальних пріоритетів та прийняття рішення

В режимі виставлення критеріальних пріорітетів та прийняття рішення (п.2.3) можливо виставити пріорітети критеріїв кількісної оцінки для даного завдання шляхом заповнення матриці попарних порівнянь критеріїв якості (див. рис. 3.30), прийняти рішення про найдоцільнішу архітектуру за експертною оцінкою по окремим критеріям (див. рис. 3.31) чи по комплексній оцінці (див. рис. 3.32). Для роботи з програмою необхідно підключитися до бази оцінок та репозитарію шаблонів, архітектур. Обрати завдання для оцінки (див. рис. 3.33), заповнити матрицю попарних порівнянь критеріїв якості (див. рис. 3.34).

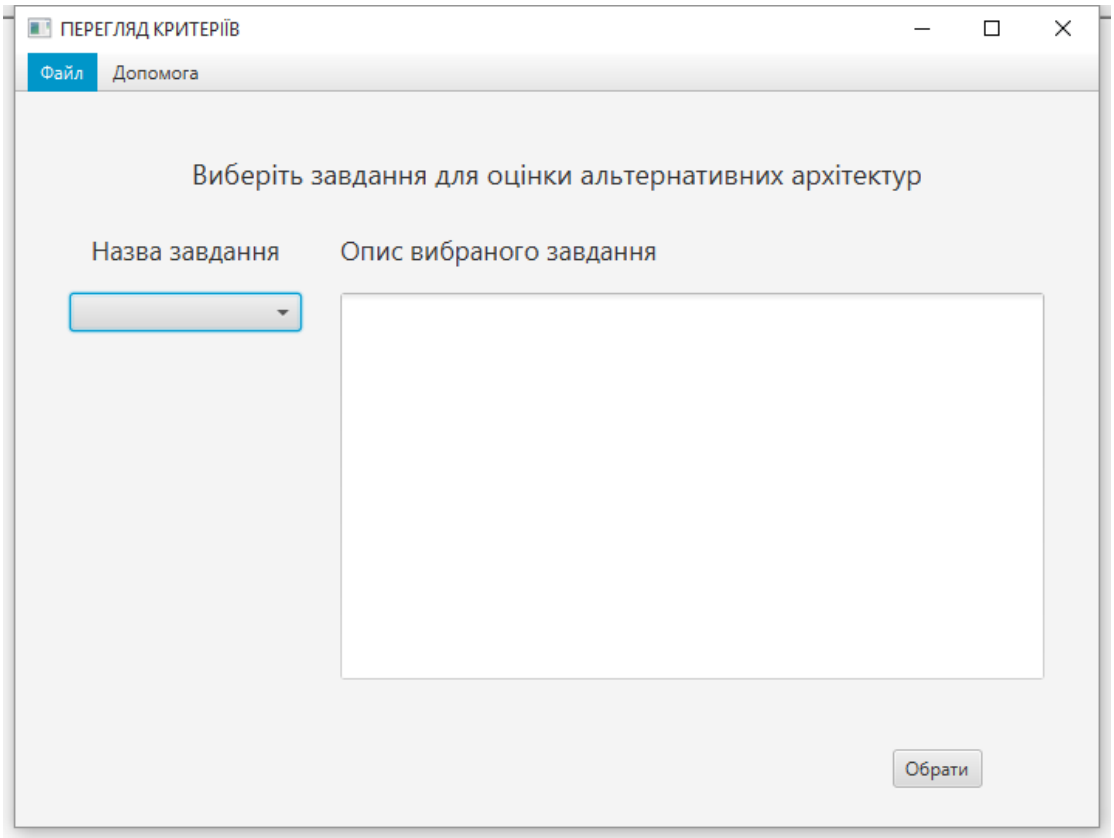


Рис. 3.30. Стартова сторінка програми виставлення критеріальних пріорітетів та прийняття рішення

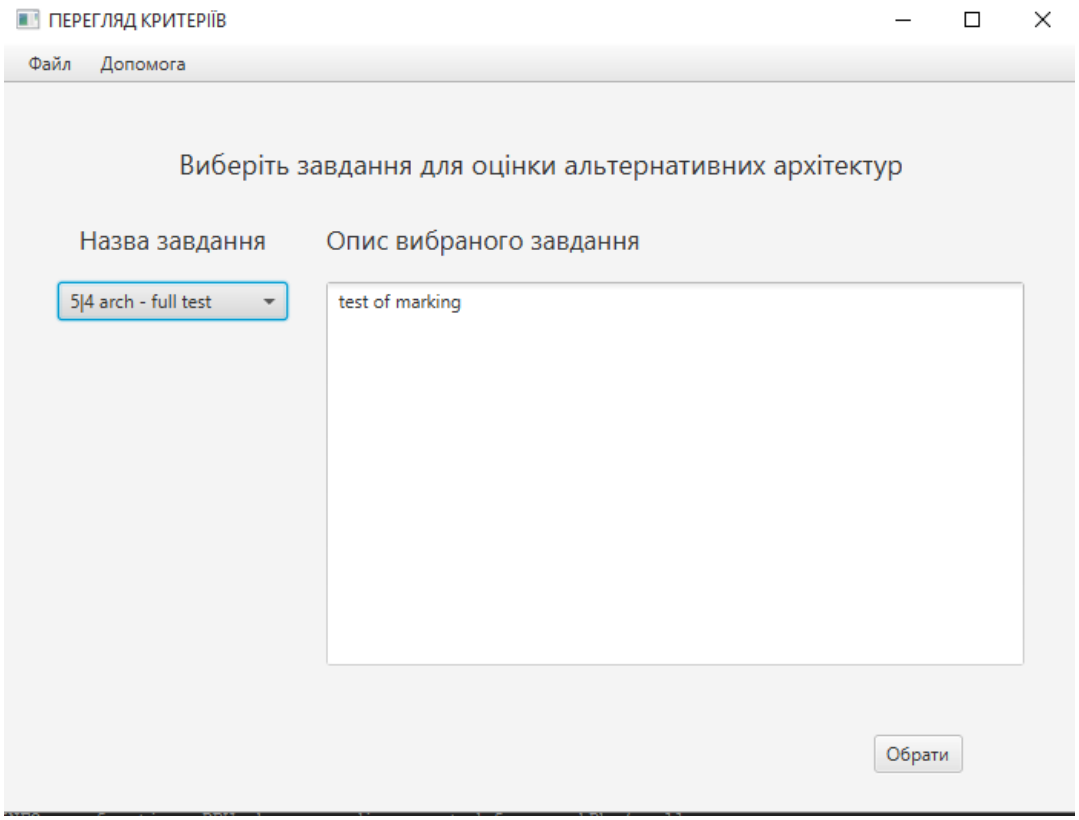


Рис. 3.31. Інтерфейс програми з вибраним завданням

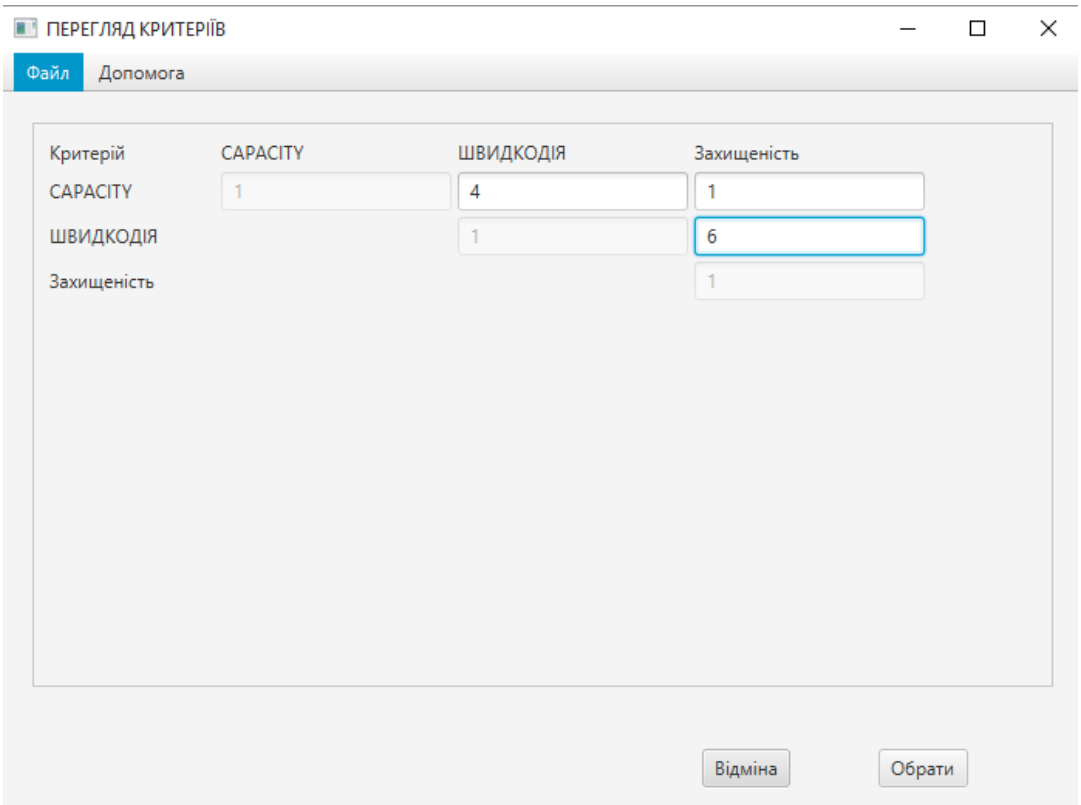


Рис. 3.32. Виставлення попарних оцінок в матрицю порівнянь критеріїв

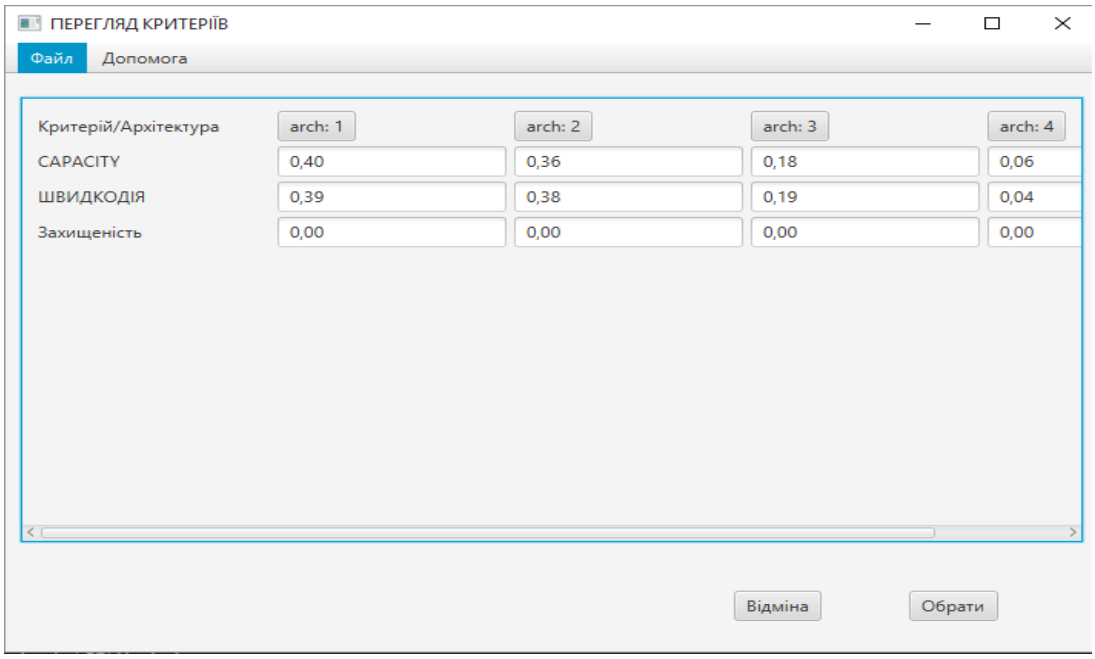


Рис. 3.33. Матриці результатів кількісної оцінки проектів архітектур

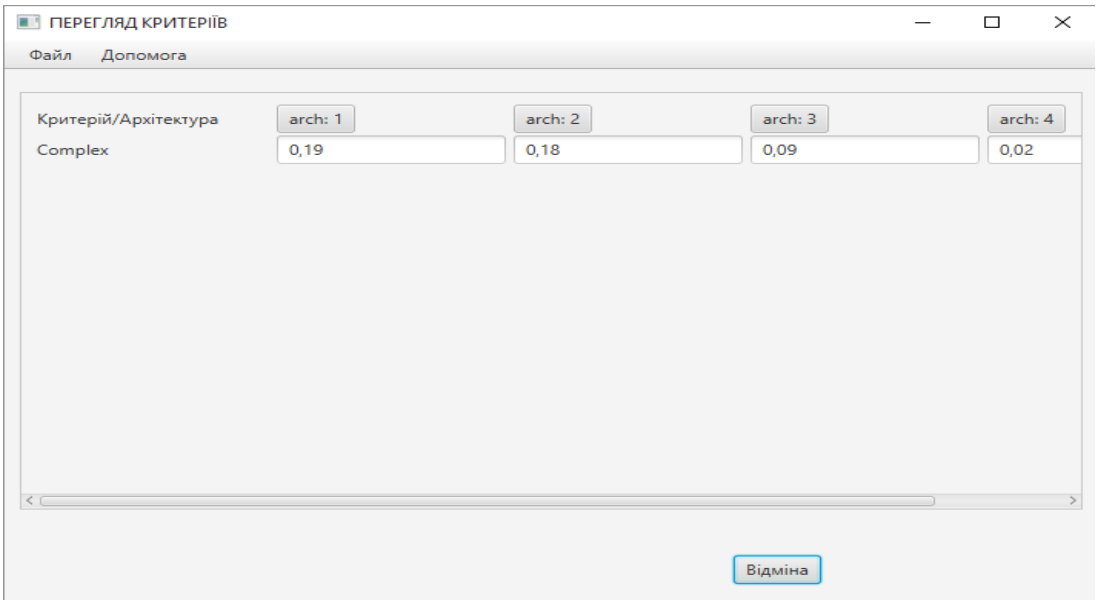


Рис. 3.34. Матриця результатів кількісної оцінки по комплексному критерію

Обрати найоптимальнішу архітектуру за результатами кількісної оцінки. В разі необхідності можна переглянути текстове чи графічне подання структури архітектури програного засобу.

РОЗДІЛ 4

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.

4.1. 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 4.1.

Таблиця 4.1

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

4.2. 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

$$З_{осн.}=Т_с*К_г, \quad (4.1)$$

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.

4.3. Розрахунок матеріальних витрат

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

$$М_{вi}=q_i*p_i, \quad (4.2)$$

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

p_i – ціна матеріалу i-го виду.

Розрахунки матеріальних витрат зведемо в таблицю 4.3

Таблиця 4.2

Зведені розрахунки витрат на оплату праці

п/п	Категорія працівників	Основна заробітна плата, грн.			Додат-кова заро-бітна плата	Нараху-вання на ФОН	Всього витрати на оплату праці, грн.
		Тарифна ставка, грн.	Кількість відпрацьо-ваних годин	Фак-тична нара-хована зар-плата в грн.			
	Керівник	6	12	72	9,20	–	–
	Кон суль-тант	5	160	800	80,00	–	–
	Техн ік	4	152	608	60,80	–	–
	Всього		320	1480	148,00	610,50	2960,50

Таблиця 4.3

Зведені розрахунки матеріальних витрат

п/п	Найменування	Од. виміру	Факт. витрачено	Ціна за одиницю, грн.	Загальна сума витрат, грн.
	Папір	пачка	3	20	60
	Компакт-диски	шт.	4	3	12
	Картридж	шт.	2	300	600
	Тонер	шт.	1	30	30
	Хостинг	місяців	12	10	120
	Разом	–	–	–	822

4.4 Розрахунок витрат на електроенергію

Витрати на електроенергію одиниці обладнання визначається за формулою

$$З_e = W*T*S, \quad (4.3)$$

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 грн.

4.5. Розрахунок транспортних затрат

$$Т_в=З_{мв}*0,08...0,1 \quad (4.4)$$

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

822,00*0,1=82,20 грн.

4.6. 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:

$$A = \frac{Б_в * Н_а}{100\%}, \quad (4.5)$$

де: A – 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 грн.

4.7. 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.

4.8. Складання кошторису витрат та визначення собівартості НДР

Результати проведених розрахунків зведемо у таблицю 4.4.

Таблиця 4.4

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 $C_в$=4123,58 грн.

4.9. Розрахунок ціни НДР

Ціна НДР визначається за формулою:

$$Ц = \frac{Св + \left(1 + P_{рен}\right) + K * B_{ні}}{K}\left(1 + ПДВ\right), \qquad (4.6)$$

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

K – кількість замовлень, од.,

$B_{ні}$ – 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.

4.10. Визначення економічної ефективності і терміну окупності капітальних вкладень

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

$$E_p = \frac{П}{C_в}, \tag{4.7}$$

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 грн.

Отримаємо Ep=2309,20/4123,58=0,56.

Термін окупності розрахуємо термін окупності розробки за формулою:

$$T_p = \frac{1}{E_p}. \tag{4.8}$$

Отже, $T_р$=1/0,56=1,8 років.

Зведені техніко-економічні показники НДР показані в таблиці 4.5.

Таблиця 4.5

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.

РОЗДІЛ 5

ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

5.1. 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.

5.2. 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.

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 5.1 (дослідження міністерства охорони здоров’я Німеччини – BGA).

Таблиця 5.1

Максимальні значення напруженості електричного поля, виміряні на відстані 50 см від екранів найбільш розповсюджених типів моніторів

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

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.). 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.

5.3. 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).

5.4. Аналіз умов праці

5.4.1. Загальна характеристика умов праці

В приміщенні бухгалтерії підприємства є місця для роботи трьох чоловік. Розміри приміщення наведені у таблиці 5.2.

Згідно СН-245-71, на одного працюючого об'єм приміщення повинен складати не менше 19,5 м^2, а площа – не менше 6 м^2.

Число працюючих у приміщенні Nр=3.

Таким чином, на кожного працюючого виходить приблизно 22,6 м^3 і 6,6 м^2, отже, усі вимоги тут дотримані.

Таблиця 5.2

Розміри приміщення

l	довжина	5 м
d	ширина	4 м
h	висота	3,4 м
S_0	площа	20 м^2
V_0	об'єм	68 м^3

Далі, відповідно до норм, повинні дотримуватися:

– ширина основних проходів, не менше: – 1200 мм

– ширина допоміжних проходів, не менше: – 700 мм

– відстань між двома столами, якщо між ними є стілець, не менше: – 1300 мм

У розглянутому приміщенні:

– ширина основних проходів: – 2000 мм

– відстань між двома комп'ютерами у ряді: – 1500 мм

Отже, норми виконуються.

5.4.2. Повітряне середовище

Шляхом провітрювання і центральної системи опалення у приміщенні бухгалтерії завжди підтримується:

– стабільна температура повітря , що становить 25 °С;

– відносна вологість повітря 55 %.

При зниженні тиску погіршується відвід тепла від елементів ЕОМ, знижуються ізоляційні властивості повітря. Показники об'єму і площі приміщення на одного працюючого відповідають нормативним значенням.

Роботи, що проводяться в бухгалтерії відносяться до легких фізичних робіт групи 1а, відповідно до ГОСТ 12.1.005-88, тому що вони проходять сидячи і не вимагають фізичного навантаження, проводяться при нормальних метеорологічних умовах і не викликають забруднення одягу і рук. Витрати енергії не перевищують 172 Дж/с (155 Ккал/год).

У таблиці 5.3 і 5.4 наведені норми температури, відносної вологості і швидкості руху повітря на робочих місцях відповідно ГОСТ 12.1.005-88, що встановлює норми виробничого мікроклімату. Дані приведені для приміщень з незначним надлишком явного тепла (до 20 Ккал/год м^3) для виконання легких робіт.

Таблиця 5.3

Норми температури, відносної вологості і швидкості руху повітря на постійних робочих місцях

Період року	Норми	Температура повітря t, ^{0}С	Відносна вологість, %	Швидкість руху повітря, м/с
холодний	оптим.	20-22	30-60	менше ніж 0,2
	доп.	17-22	менше 75	менше ніж 0,3
теплий	оптим.	20-25	30-60	0,2-0,3
	доп.	менше ніж 28	менше ніж 80*	0,3-0,5
* - у теплий період року припустима відносна вологість повітря для всіх приміщень і категорій робіт має значення, приведені в таблиці 5.4				

Таблиця 5.4

Відносна вологість повітря в теплий період року

Температура повітря,	28	27	26	25	2	<=
Відносна вологість, %	>=	60	65	70	7	75

Основними джерелами тепла в приміщенні є:

- сонячна радіація;
- система опалення;
- люди, що працюють у приміщенні;
- устаткування.

У таблиці 5.5 наведені дані вимірювання в приміщенні ОЦ у місяці грудні.

Таблиця 5.5

Результати виміру параметрів мікроклімату в бухгалтерії

Температура повітря t, °C	20 – 25
Відносна вологість, %	50 – 60
Швидкість руху повітря, м/с	0,2

Як видно з таблиці 5.5, у розглянутому приміщенні значення параметрів мікроклімату відповідають нормативним. Стабільність цих параметрів підтримується загальною системою утеплення і кондиціонування повітря.

5.4.3. Освітлення

У приміщенні бухгалтерії використовується природне і штучне освітлення. Природне освітлення здійснюється за допомогою двох вікон загальною площею S=7,5 м^2, що забезпечує коефіцієнт природної освітленості E=1,5%. Це відповідає СНиП I-4-79.

Штучне освітлення в бухгалтерії здійснюється системою загального рівномірного освітлення, що реалізована на основі люмінесцентних ламп типу ЛДЦ-40-1, які мають наступні параметри:

– висока світловіддача;
– тривалий термін служби;
– мала яскравість освітлювальної поверхні;
– близькість спеціального складу до природного освітлення.

Робота за монітором ПЕОМ по розряду зорових робіт відноситься до III типу (роботи високої точності з розміром об'єкта 02-0,4 мм). При загальному освітленні, освітленість робочого місця повинна складати від 200 до 400 лк.

При штучному освітленні нормуються наступні параметри:

- *E* (лк) - найменша припустима освітленість;
- *M* - показник дискомфорту;
- *Кп* (%) - коефіцієнт пульсації освітлення.

Перевіримо відповідність фактичних параметрів штучного освітлення в приміщенні нормам. Номінальний світловий потік лампи білого свічення ЛДЦ-40-1: *Фл* = 3120 лм.

У приміщенні застосовуються світильники, у яких встановлені дві лампи. Висоту підвіски світильника визначимо по формулі:

$$h = H - h_C - h_P - h_П, \qquad (5.1)$$

де:

H – висота приміщення, м.;

h_C – висота світильника, м.;

$h_П$ – відстань від стелі до підвіски, м.;

h_P – висота робочої поверхні, м.;

Для розглянутого приміщення:

H = 3,4 м,

h_C = 0,15 м,

$h_П$ – 0 м, (підвісу немає)

$h_Р$ = 0,8 м.

Ззвідси h = 3,4 – 0,15 – 0,8 = 2,45м.

Світильники розташовані в 2 ряди. Висота підвіски світильників складає 2,45 метра відносно підлоги, відстань між рядами 1 м, відстань від ряду до стіни 1,5 метра. Приміщення має наступні габарити:

- довжина A = 5 метрів;
- ширина B = 4 метрів.

Визначимо освітленість у робочій точці. Для розрахунку загальної рівномірної освітленості при горизонтальній робочій поверхні використовуємо метод коефіцієнта використання світлового потоку.

Розрахункова формула для світлового потоку світильника має такий вигляд:

$$Фл = \frac{E \cdot Кз \cdot S \cdot Z}{N \cdot n}, \tag{5.2}$$

де N - кількість світильників у приміщенні, $N = 3 \cdot 2 = 6$;

n - коефіцієнт використання світлового потоку;

$Фл$ - світловий потік ламп;

$Кз$ - коефіцієнт запасу, $Кз$ = 1,5;

Z - коефіцієнт нерівномірності;

S - площа приміщення;

E - освітленість, створювана усіма світильниками.

Звідси одержуємо формулу для розрахунку освітленості на робочому місці:

$$E = \frac{Фл \cdot N \cdot n}{Кз \cdot S \cdot Z} \tag{5.3}$$

Коефіцієнт використання світлового потоку залежить від:

- ККД кривої розподілу сили світла світильника;

- коефіцієнта відбивання стелі $R_П$ і стін R_C,;
- висоти підвісу світильників $h_П$;
- показника приміщення i:

$$i = \frac{A \cdot B}{h \cdot (A + B)} \quad . \tag{5.4}$$

i = (5 • 4) / (2,45 • (5 + 4)) = 0,408

Стеля і стіни пофарбовані в білий колір.

Приймаємо:

$R_П$ = 50%

$R_{C,}$ = 30%.

Звідси:

n = 31%.

$$E = \frac{(3120 \cdot 2) \cdot 6 \cdot 0{,}31}{20 \cdot 1{,}1 \cdot 1{,}5} = 352 лк$$

Так як по розряду зорової роботи робота за дисплеєм ПЕОМ відноситься до III типу (високої точності, розмір об'єкта 0.2-14 мм), то при загальному висвітленні освітленість робочого місця повинна складати від 200 до 400 лк, рекомендована освітленість при роботі з дисплеєм ПЕОМ складає 200 лк, а при сполученні роботи з документами 400 лк. Фактична освітленість на робочому місці складає 352 лк.

Таким чином для роботи з дисплеєм цілком достатньо існуючих джерел світла, однак робота з документами повинна вестися при природному освітленні, або за допомогою додаткових місцевих джерел освітлення.

5.4.4. Шум

З фізіологічної точки зору шумом є всякий небажаний, неприємний для сприйняття людини шум. Шум погіршує умови праці,

впливаючи на організм людини. При тривалому впливі шуму на організм людини відбуваються небажані явища:

- знижується гострота зору, слуху;
- підвищується кров'яний тиск;
- знижується увага.

Сильний тривалий шум може бути причиною функціональних змін серцево-судинної і нервової систем, що приводить до захворювань серця та підвищеної нервозності.

Характеристикою постійного шуму на робочих місцях є рівні звукового тиску в Дб у октавних смугах із середньогеометричними частотами 31,5, 63, 125, 250, 500, 1000, 2000, 4000, 8000 Гц. Припустимим рівнем звукового тиску в октавних смугах частот, рівні звуку і еквівалентні рівні звуку на робочому місці варто приймати дані з таблиці 5.6.

Таблиця 5.6

Припустимі рівні звукового тиску

Робоче місце	Рівні звукового тиску в дБ, в октавних смугах із середньогеометричними частотами в Гц								Рівні звуку в еквівалентних рівнях звуку в дБ
	63	125	250	500	1000	2000	4000	8000	
Операторів, програмістів	71	61	54	49	45	42	41	38	50

5.4.5. Випромінювання електромагнітних полів

Електромагнітні випромінювання низької частоти (від 12 до 150 Гц) роблять найбільш шкідливий вплив на організм людини Тривалий

вплив низькочастотних полів сприяє порушенню репродуктивної функції і виникненню раку.

Для зниження рівня перемінного електромагнітного поля в сучасних моніторах, що відповідають специфікаціям Low Radiation (LR), MPRII і TCQ92, застосовуються котушки компенсації, встановлені на електронно-променевій трубці (ЕПТ), а також спеціальні матеріали в її конструкції.

Застосовувані, при роботі в приміщенні бухгалтерії монітори задовольняють встановлені норми.

5.4.6. Електробезпека

Електричний струм здійснює на організм людини електролітичний, термічний і біологічний вплив. Викликає загальну рефлекторну реакцію нервової і серцево-судинної системи. Результат електротравми залежить як від умов зовнішнього середовища так і від параметрів організму людини. Ступінь поразки людини залежить від роду і величини напруги і струму, частоти електричного струму, шляху струму через людину, тривалості дії й умов зовнішнього середовища.

Для запобігання пораження людини електричним струмом застосовують захисне заземлення. Захисним заземленням називається навмисне електричне з'єднання з землею чи її еквівалентом металевих не струмоведучих частин, що можуть виявитися під напругою.

Пристрій, що заземлює, складається з одного чи декількох заземлюючих, металевих елементів занурених на визначену глибину в ґрунт і провідників, що заземлюють, з’єднуюче устаткування, що заземлюється, із заземлювачем. Принцип дії захисного заземлення заснований на зниженні напруги щодо землі до припустимих рівнів напруги дотику.

Згідно правил будови електроустановок при напрузі до 1000 В опір пристрою, що заземлює, повинне бути не більш 4 Ом, при напрузі понад 1000 В опір повинний бути не більш 10 Ом.

РОЗДІЛ 6

ЕКОЛОГІЯ

6.1. Актуальність екологічної проблеми

Дипломна робота стосується розробки програмонго 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 існукють 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, самртфони тощо. Дія їхня аналогічна до впливу ПК на довкілля. Тому проаналізуємо вплив електромагнітого випромінювання цих пристроїв та заходи, спрямовані на зменшення цього впливу.

6.2. Основні джерела забруднення, що створює технічний об’єкт

Проблема електромагнітного випромінювання від персонального комп’ютера достатньо гостра всилу 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 піддється 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.

6.3. 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 900 – 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.

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

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.

App	Layer *i*	Category	Pattern *i*

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. Ящишин // 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.

УДК 004.415.5
Я. Самиця[1], М. Горалечко[1], Ю. Дзиґа[2]
[1]Тернопільський національний технічний університет імені Івана Пулюя
[2]Національний технічний університет України "Київський політехнічний інститут імені Ігоря Сікорського"

ІЄРАРХІЧНА СТРУКТУРА МОДЕЛЕЙ ЯКОСТІ СИСТЕМ ЕЛЕКТРОННОЇ КОМЕРЦІЇ

UDC 004.415.5
Ya. Samytsia[1], M. Horalechko[1], Yu. Dzyha[2]
[1](Ternopil Ivan Puluj National Technical University, Ukraine)
[2](National Technical University of Ukraine "Kyiv Sikorsky Polytechnic Institute", Ukraine)

HIERARCHIC STRUCTURE FOR E-COMMERCE SYSTEMS QUALITY MODELS

Для розподілення встановленого цільового значення рівня надійності за компонентами програмних систем (ПС), які відносяться до класу платформ електронної комерції, запропоновано чотирирівневу ієрархічна структуру, кожний рівень якої відповідає рівню бачення проблеми якості відповідною категорією учасників проекту ПС, а саме:

- замовника, який зацікавлений у загальній якості ПС при її використанні ($Qпс$);
- користувачів, які пов'язують загальну якість системи з надійним виконанням множини функцій ПС ($F_1, ..., F_k$);
- менеджерів (та аналітиків), які пов'язують надійне виконання кожної функції F_i з надійною роботою множини розроблюваних програмних застосувань ($Z_1, ..., Z_l$), призначених для автоматизованої підтримки функцій;
- проектувальників, які пов'язують надійність кожного програмного засобу Z_i з надійністю множини розроблюваних, а також повторно використовуваних незалежних модулів ($M_1, ..., M_n$). Припущення незалежності відповідає сучасним концепціям об'єктно-орієнтованого та компонентного підходів до розроблення ПС.

Ієрархічна декомпозиція є природним засобом спрощення проблеми в системах оброблення даних, не пов'язаних з функціонуванням в реальному масштабі часу. Вона властива сучасним CASE-технологіям, які застосовуються для побудови ПС.

Приклад ієрархічної структури ПС подано на рис. 1.

Основна мета побудови і аналізу ієрархії ПС – отримати параметри моделі розподілу надійності на кожному її рівні з урахуванням важливості компонентів кожного з рівнів 2 – 4 для загальної якості платформ електронної комерції.

Для визначення ваги окремих компонентів у ієрархії пропонується використати метод аналізу ієрархій (МАІ) [1, 2]. За цим методом визначаються:

1) вектори локальних пріоритетів функцій, програмних засобів та модулів, а саме:

$U = (u_1, u_2, ..., u_k)$ –вектор коефіцієнтів відносної ваги функцій у $Q_{пс}$;

$V_i = (v_{i1}, v_{i2}, ..., v_{il})$, i = *1*, ...k –вектори коефіцієнтів відносної ваги програмних застосувань для кожної функції;

$W_i = (w_{i1}, w_{i2}, ..., w_{in})$, i = *1*, ...l – вектори коефіцієнтів відносної ваги модулів для кожного програмного застосування;

2) вектори загальних (глобальних) пріоритетів програмних застосувань та модулів. Загальна вага i-го програмного засобу розраховується за формулою $V^*_i = \sum_{j=1}^{k} u_j \cdot v_{ij}$

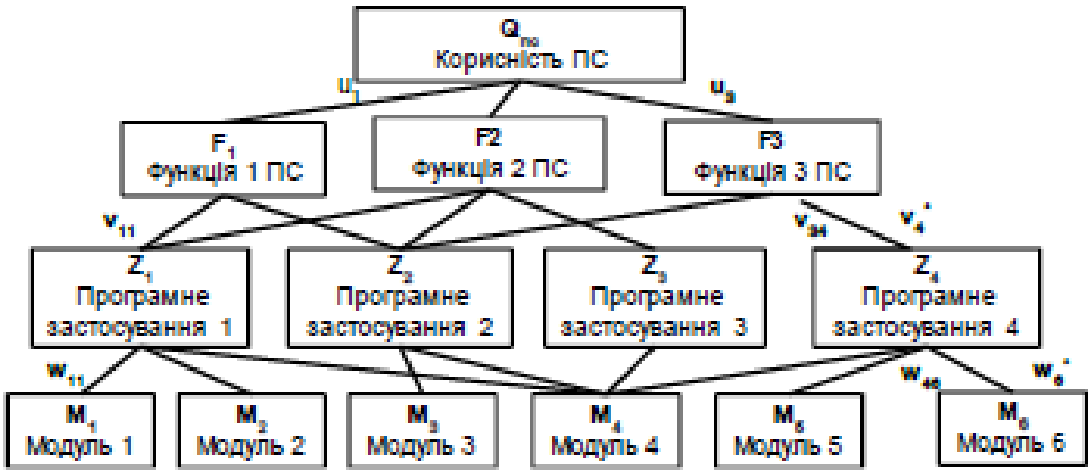


Рисунок 1 – Чотирирівнева ієрархічна структура ПС

Для всіх програмних застосувань та по відношенню до всіх функцій вектор загальних вагових коефіцієнтів визначається так:

$$V^* = U \cdot \begin{pmatrix} V_1^* \\ V_2^* \\ \dots \\ V_l^* \end{pmatrix} \quad \text{або} \quad (v_1^* \quad v_2^* \quad \dots \quad v_l^*) = (u_1 \quad u_2 \quad \dots \quad u_k) \begin{pmatrix} v_{11} & v_{12} & \dots & v_{1l} \\ v_{21} & v_{22} & \dots & v_{2l} \\ \dots & \dots & \dots & \dots \\ v_{k1} & v_{k2} & \dots & v_{kl} \end{pmatrix} \quad (1)$$

Так само визначається вектор загальних вагових коефіцієнтів для всіх модулів, до яких є звернення у програмних застосуваннях:

$$(w_1^* \quad w_2^* \quad \dots \quad w_m^*) = (v_1^* \quad v_2^* \quad \dots \quad v_l^*) \begin{pmatrix} w_{11} & w_{12} & \dots & w_{1m} \\ w_{21} & w_{22} & \dots & w_{2m} \\ \dots & \dots & \dots & \dots \\ w_{l1} & w_{l2} & \dots & w_{lm} \end{pmatrix} \quad (2)$$

Отримані загальні вагові коефіцієнти для множини функцій, програмних застосувань та модулів далі використовуються при розробці методу оцінювання якості платформ електронної комерції. Кожний з цих вагових коефіцієнтів є оцінкою ступеню важливості надійної роботи відповідного компоненту інтернет-магазину для забезпечення його загальної експлуатаційної якості за критерієм надійності.

Література

1. Alexandr Harchenko. DecisionSupportSystemofSoftwareArchitect // Alexandr Harchenko, Ihor Bodnarchuk, Iryna Halay // Proceeding of the 2013 IEEE 7th International Conference on Intelligent Data Acquisition and Advanced Computing Systems (IDAACS). Volume 1, pp. 265–269, Berlin.
2. Saaty T. Decision Making with the Analytic Network Process. / Saaty T. Vargas L. // – N. Y.: Springer, 2006. 278 p.

СЕКЦІЯ 2. ІНФОРМАЦІЙНІ СИСТЕМИ ТА ТЕХНОЛОГІЇ