ЗМІСТ

ВСТУП

Актуальність теми. Сучасні інформаційні технології відіграють важливу роль в житі сучасної людини. Новітні засоби комунікації, апаратні системи, інтелектуальне виробниче обладнання, високопродуктивні програмні системи – усі ці компоненти застосовуються для підвищення показників ефективності виробничих і технологічних процесів. Однак широке впровадженння інформаційних технологій безпосередньо зумовило зростання складності програмних систем. Складність розробки програмних систем полягає у недосконалості формальних методів і технологій проектування, не достатній повноті та адекватності відображення бізнес процесів щодо збору, обробки, надійного зберігання та представлення різного роду інформації. Як наслідок такі фактори негативно відображаються на якості кінцевих програмних продуктів. Раніше проблеми складності вирішувалися розробниками шляхом вибору оптимальних, з їх точки зору, структур даних, розробки алгоритмів і застосування концепції розмежування повноважень. Особливо важливим є забезпечення якості на етапі проектування архітектури програмних систем, оскільки архітектура є базою для розгортання та реалізації як функціональних, так і не функціональних вимог до програмних систем.

Хоча термін «архітектура програмного забезпечення» є відносно новим для індустрії розробки ПЗ, фундаментальні принципи цієї області невпорядковано застосовувалися при розробці ПЗ починаючи з середини 80-х років. У 90-ті роки розпочинаються спроба визначити і систематизувати основні аспекти даної дисципліни такі як: початковий набір шаблонів проектування, стилів дизайну, мов опису і формальна логіка. Основною ідеєю програмної архітектури є зниження складності системи шляхом абстракції і розмежування повноважень. Але проектування архітектури ПЗ до цього часу є складним процесом, який базується на знаннях і досвіді системних аналітиків. Складність проектування архітектури ПЗ полягає в тому, що будь-яка система передбачає наявність області застосування і цілей замовника. В залежності від ключових цілей реалізації системи, архітектуру

можна опиати за допомогою сценаріїв та нефункціональних вимог. Нефункціональні вимоги (атрибути якості) системи включають в себе відмовостійкість, збереження зворотної сумісності, розширюваність, надійність, придатність до сервісного обслуговування, доступність, безпека, зручність використання, а також інші властивості. Загалом існує чотири основних види проектування архітектури програмного забезпечення – це стандартизований підхід, об'єктний підхід, загальносистемний підхід і шаблонний підхід або патерний. Шаблони проектування на даний час є ефективним способи вирішення задач проектування архітектури програмного забезпечення. За рахунок шаблонів проводиться уніфікація термінології, назв модулів і елементів проекту. Правильно сформульований шаблон проектування дозволяє знайти оптимальне рішення, яке в подальшому може бути повторно використане.

Замовник передбачає цілі ПЗ і визначає вимоги до ПЗ. Саме задоволення цих вимог визначає якість програмного забезпечення. Дослідженню якості програмних систем, присвячено ряд наукових та науково-прикладних публікацій, як українських науковців (Г. Мороз, Г. Коваль, Т Коротун, О. Харченко), так і закордонних (I. Sommervill, E. Брауде, В. Липаев, G. Mayers, M. Holsted). У цих роботах пропонується ряд методів та моделей, що дають змогу підвищити адекватність відображення потреб замовника на реалізацію властивостей ПЗ, врахувати та забезпечити ряд додаткових характеристик якості. Також багато наукових публікацій присвячено дослідженню архітектури програмного забезпечення, серед яких варто виділити праці Ф. Крачтена, Х. Оббінка, Д. Стаффорд, Г. Коваль, Е. Брауде, К. Чернецки. Однак комплексного підходу щодо побудови або вибору оптимальної архітектури ПЗ, який би одночасно враховував вимоги до ПЗ та критерії якості архітектури на даний час ще не запропоновано. Тому актуальною науково-технічною задачею є обґрунтування методу і засобу для аналізу вимог до ПЗ та оцінювання якості патернів архітектур, які дали б змогу приймати оптимальні рішення при проектуванні архітектури та враховували критерії повноти і адекватності відображення вимог на архітектуру ПЗ.

Зв'язок магістерської роботи з науковими програмами, планами, темами. Робота виконана на кафедрі комп’ютерних системи та мереж Тернопільського національного технічного університету імені Івана Пулюя в рамках науково-дослідницької роботи ДІ 207-13 "Розробка, дослідження та впровадження методів і засобів контролю та управління якістю програмних продуктів", номер держреєстрації №0113U000258.

Мета роботи: дослідити та обґрунтувати методи і засоби аналізу вимог та оцінювання якості патернів архітектур програмного забезпечення.

Основні задачі дослідження:

1. Провести аналіз наукових публікацій та стандартів галузі інженерії програмного забезпечення для визначення сучасного стану інтеграції процесів забезпечення якості ПЗ на стадіях ЖЦ.

2. Дослідити та обґрунтувати моделі представлення вимог до архітектури ПЗ для забезпечення повноти і адекватності вибору патернів в проектуванні.

3. Розробити формалізований метод і процедуру детектування шаблонів для забезпечення оптимальної побудови архітектури ПЗ.

4. Створити та впровадити інструментальні програмні засоби для автоматичного вибору архітектурних патернів відповідно до заданих вимог.

Об’єкт дослідження – процеси забезпечення якості вимог та архітектури ПЗ.

Предмет дослідження – моделі, методи і засоби забезпечення та оцінювання якості патернів архітектури ПЗ.

Наукова новизна отриманих результатів. Наукова новизна полягає у вирішенні науково-практичної задачі методів і засобів аналізу вимог та оцінювання якості патернів архітектур програмного забезпечення, при цьому одержано наступні результати:

– уперше, обґрунтовано застосування апарату нейронних мереж для визначення вимог до архітектурних патернів, що дало змогу більш повно, в порівняні з іншими апаратами, відобразити вимоги до ПЗ на його архітектуру.

– уперше запропоновано метод детектування архітектурних патернів програмного забезпечення на основі рекомендацій стандарту ISO 9126 та нейронних мереж, що дали змогу побудувати комунікацію вимог між архітектурою «високого рівня» та «модульною» архітектурою.

– уперше розроблено засоби для автоматичного вибору архітектурних патернів відповідно до заданих вимог з використанням нейронної мережі NeuronDotNet, що дало змогу підвищити ефективність та гнучкість етапу проектування архітектури ПЗ.

Практичне значення одержаних результатів. Впровадження методу і автоматизованої системи вибору архітектурних патернів дає змогу інтегрувати процес управління якістю ПЗ у технологію проектування, і таким чином зменшити витрати на повторне виконання робіт, спричинене невиконанням вимог якості ПЗ, а також зменшити ризик відхилення проекту замовником.

Інструментальні програмні засоби вибору архітектурних шаблонів впроваджено у ТОВ «ДРІМС АЙТИ». Акти впровадження у виробництво наведено у додатку А.

Публікації. Результати дослідження апробовано на науково-практичній конференції Тернопільського національного технічного університету імені Івана Пулюя у вигляді тез конференцій.

РОЗДІЛ 1
СУЧАСНИЙ СТАН ДОСЛІДЖЕНЬ В ОБЛАСТІ ЗАБЕЗПЕЧЕННЯ ЯКОСТІ ПРОГРАМНИХ СИСТЕМ НА ЕТАПИ ЖЦ

У даному розділі розглянуто підходи до розробки ПЗ та проведено аналіз методів забезпечення та контролю якості проекту на різних стадіях ЖЦ, визначено основні фактори негативного впливу на якість ПЗ, поставлено задачі, які необхідно розв’язати в процесі проектування для задоволення потреб користувачів у ПЗ, розглянуто моделі якості ПЗ та здійснено їх порівняння.

1.1 Методи забезпечення якості ПЗ

Сучасний етап розвитку інженерії програмного забезпечення характеризується створенням і застосуванням ряду методологій, які спрямовані на прискорення процесу розробки ПЗ. При цьому розробники часто не достатньо уваги приділяють забезпеченню якості. Так, за статистикою, в США тільки близько 53% програмних проектів завершується вдало, а з них лише в 42% забезпечуються всі вимоги, в тому числі і вимоги до якості [1].

Питання високої конкурентоспроможності програмних продуктів, зниження ризиків проектів, досягнення балансу між тривалістю, вартістю та якістю проекту, протягом останніх десятиріч є доволі актуальним не тільки для розробників ПЗ, а й для замовників, і користувачів. Це підтверджується рядом наукових публікацій [2,3] у яких досліджено фактори впливу на якість ПЗ та запропоновано шляхи мінімізації ризиків при виконанні проекту.

Одним із ефективних шляхів вирішення цих задач є застосування методологій, які базуються та реалізується через конкретні технології підтримки процесів контролю та забезпечення якості ПЗ на стадіях ЖЦ ПЗ. Технологічні процеси управління, контролю та забезпечення якості на стадіях ЖЦ визначені стандартами [5]. Однак для виконання цих процесів необхідно розробити

автоматизовані інструментальні засоби їх підтримки.

Важливість процесу забезпечення якості ПЗ обумовлено також і наявністю великої кількості технологій розробки ПЗ, що дозволяє спроектувати їх різними способами, але при цьому виникає проблема, пов’язана з об’єктивністю та адекватністю методів досягнення якості на стадіях ЖЦ ПЗ та оцінювання якості кінцевого програмного продукту. Це пояснюється використанням розробниками своїх корпоративних технології та критеріїв оцінювання якості, які часто є неузгодженими і не стандартизованими.

У переважної більшості розробників ПЗ висока якість асоціюється із здатністю задовольняти функціональні вимоги, які визначені з потреб користувача. Однак якісна реалізація заданих функціональних властивостей не забезпечує і часто залишає поза увагою вимоги до якості, таких як зручності, швидкодії, надійності. Також дані не функціональні вимоги часто пов’язані з архітектурою програмного забезпечення, що є не видимою для кінцевого користувача. Вирішити ці проблеми можна шляхом вдосконалення існуючих методів контролю і забезпечення якості ПЗ, а саме розробити інструмент для підтримки розробки архітектури ПЗ.

Так, І. Соммервіл, як науковець та автор багатьох праць [6] у сфері інженерії програмного забезпечення зробив значний внесок у стандартизацію процесів, що пов’язані з контролем та управлінням якістю ПЗ. У [6] проаналізовано ряд методологій, методів і засобів розробки ПЗ, проведено їх порівняння та доведено важливість забезпечення та оцінювання якості у загальному процесі проектування ПЗ. Однак у [**Ошибка! Источник ссылки не найден.**] пропонується використання рекомендацій стандартів [7,8,9,10] для оцінювання якості ПЗ, при цьому немає чітко вираженої та науково-обгрунтованої стратегії її проведення. Відсутні рекомендації аналізу атрибутів, методів їх оцінювання та комплексної інтегрованої інтерпретації якості ПЗ. Контроль відповідності вимог до ПЗ на різних стадіях ЖЦ проводиться шляхом тестування, але технологій їх комунікації не наведено. У результаті відсутності формалізованої процедури комунікації вимог можлива втрата зв’язку

між відповідними вимогами та їхніми трансформаціями на різних стадіях розробки, що може призвести до значного погіршення якості ПЗ.

На практиці для забезпечення якості ПЗ розробники використовують організаційні та технологічні заходи. Організаційні заходи щодо контролю та забезпечення якості ПЗ базуються на використанні людських ресурсів для моніторингу процесів розробки проекту. При цьому передбачається проведення інспекції та рев'ю коду, тестування та ряду інших заходів. Однак такий підхід вимагає значних трудових та економічних затрат і результати використання таких заходів не завжди адекватно відображають реальний стан якості ПЗ.

Технологічні заходи щодо забезпечення якості ПЗ передбачають впровадження технологічних методів контролю якості, пов'язаних із особливостями технології розробки проекту. Серед технологічних підходів можна виділити два основних підходи до забезпечення якості ПЗ: продукто-орієнтований – підхід, який реалізується шляхом контролю якості готового програмного продукту; процесо-орієнтований – підхід, в якому контролюється якість процесів створення ПЗ на етапах ЖЦ.

Перший підхід базується на визначенні та застосуванні правил класифікації, специфікації та оцінювання характеристик якості готових системних продуктів. При цьому якість ПЗ контролюється на завершальних етапах проектування і використовується для атестації та сертифікації одиничних програмних проектів, які реалізуються невеликою серією. Проте, коли виявляється не відповідність властивостей ПЗ заявленим у специфікації вимогам, то це відображається у значні додаткові вартісні затрати на проект та збільшення терміну його виконання.

Перші спроби розробки методів оцінювання якості ПЗ запропоновано в [11]. Суть цього методу полягає у визначенні властивостей ПЗ і встановленні за допомогою формальних операторів відповідних їм метрик. Однак оцінювання якості готового програмного продукту дозволяло лише виявляти не реалізовані або частково реалізовані потреби у ПЗ. При цьому рекомендації щодо покращення якості відображались не завжди об'єктивно, оскільки критерії, згідно яких

проводилось оцінювання, носили суб'єктивний характер і не були загальноприйнятими. Крім того, не досліджувався вплив одного критерію якості на інші. В ряді випадків покращення одного показника призводив до погіршення одного або декількох інших. Проте на той час це був значний прогрес, оскільки розробники почали звертати увагу на важливість забезпечення якості ПЗ як на етапах розробки проекту, так і на етапі оцінювання їх якості..

У роботі [12] досліджено методи та моделі забезпечення якості ПЗ з точки зору надійності. Запропонована автором методика базується на рекомендаціях стандарту ISO 9126-1. Однак потрібно зазначити, що процедуру застосування методу забезпечення як надійності, так і якості ПЗ в цілому на практиці втілити досить складно. Виходячи з цього, не в повній мірі відображається якість ПЗ з точки зору користувача, оскільки враховуються тільки безвідмовність роботи програмного продукту.

Процесо-орієнтований підхід до проектування ПЗ передбачає контроль та визначення стану виконання процесів на стадіях ЖЦ. При цьому, результат використання такого підходу не забезпечує в повному обсязі виконання усіх вимог до ПЗ, оскільки методи контролю і оцінювання якості, на яких базуються технології реалізації процесо-орієнтованого підходу, недостатньо формалізовані та не уніфіковані.

Так у роботах [13, 14] для забезпечення належної якості виконання процесів на стадіях ЖЦ, запропоновано динамічні процедури тестування, які дають можливість виявляти помилки на заданих тестових наборах даних. Однак такий підхід має і ряд недоліків, основний з них полягає у відсутності процедури перевірки фактичних значень критеріїв якості нормованим показникам. Крім того, застосувати методи тестування на кожній стадії ЖЦ надто складно, оскільки це потребує створення формалізованого апарату комунікації критеріїв якості.

Технології, наведені у [14] базуються на проведенні інспекції процесів для визначення стану їх якості. Проте методи та процедури їх реалізації є

неформалізованими і спостерігається суттєвий вплив суб'єктивних факторів, пов'язаних з недостатньою кваліфікацією експертів.

На практиці широко використовують наведені вище технології та методи забезпечення, контролю та управління якістю, проте як продукто-орієнтований, так і процесо-орієнтований підходи мають суттєві недоліки і потребують подальшого вдосконалення методів і технологій їх реалізації. Про це свідчать результати досліджень, які наведені на рис. 1.1. З аналізу цих результатів випливає, що неякісне виконання проектів є причиною суттєвих економічних збитків, а отже сприяє зростанню ризиків при виконанні проекту [1].

Найбільш оптимальним шляхом вирішення проблем, пов'язаних із забезпеченням якості ПЗ, є поєднання принципів продукто-орієнтованого та процесо-орієнтованого підходів. При цьому технологія реалізації такого підходу повинна враховувати комунікацію вимог до якості ПЗ на всіх стадіях ЖЦ та забезпечувати контроль їх виконання, оскільки це дозволить виявляти невідповідності вимогам на етапах виконання проекту і своєчасно вносити необхідні зміни, що значно скоротить витрати на доопрацювання проекту і термін його завершення.

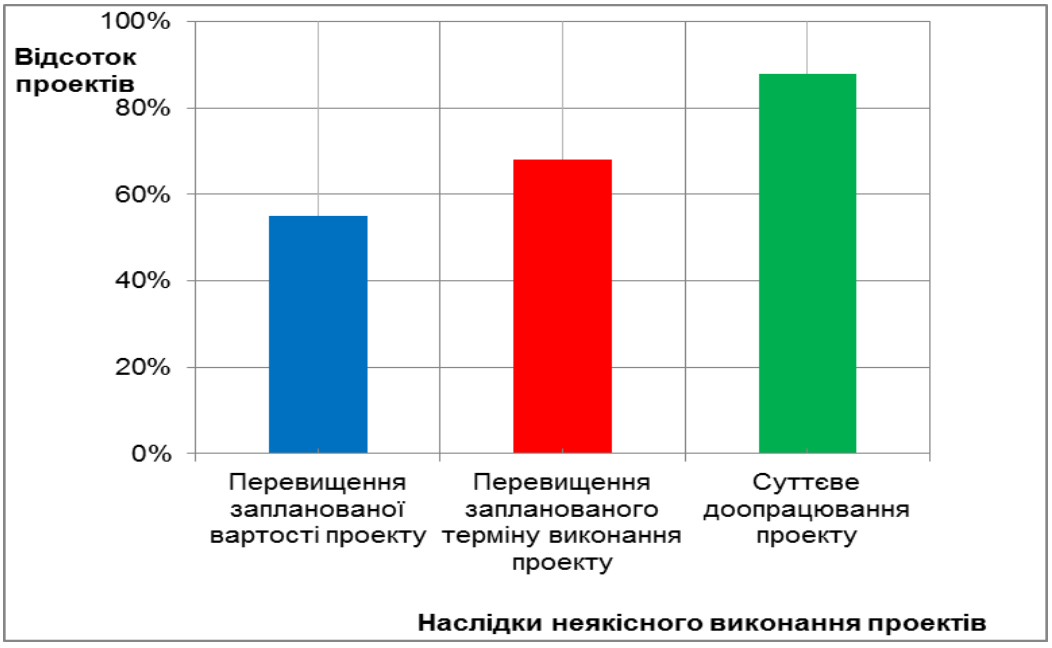

Рис. 1.1 – Наслідки неякісної реалізації проектів

Застосувати поєднання продукто- і процесо-орієнтованих підходів можна тоді, коли розробники ПЗ використовують моделі ЖЦ із зворотніми зв'язками. Це обумовлено необхідністю використання ітераційних процедур контролю якості на кожному етапі з можливістю повернення до попередніх.

Практичне впровадження і застосування технологій, аналіз і порівняння підходів розробки ПЗ та системна концепція побудови архітектури ПЗ запропонована у [15]. Однак ця концепція не базується на уніфікованих і структурованих моделях і технологіях розробки ПЗ. У [15] при проектуванні ПЗ використовується ітераційний підхід, при цьому забезпечується контроль якості виконання процесів на стадіях життєвого циклу. Контроль якості проводиться шляхом перевірки (тестування) вхідних критеріїв відносно вихідних на кожній із стадій. Даний підхід не дозволяє оцінювати властивості власне самої ПЗ, оскільки відсутня методологія узгодження та відображення якості процесів на етапах проектування. У результаті застосування аналітичних методів для перетворення показників якості процесів, отримують суб'єктивні, не уніфіковані та не систематизовані критерії якості.

Розглянемо технології проектування ПЗ на стадіях ЖЦ та проведемо аналіз процесів забезпечення якості, які реалізованих у них. Досить важливою з цієї точки зору є стадія розробки вимог до ПЗ, оскільки, вимоги є базою для планування та розробки проекту, управління ризиками, атестації та сертифікації системи. На цій стадії визначено декілька послідовно виконуваних процесів:

- визначення та аналіз вимог;
- представлення вимог;
- трасування вимог;
- розробка процедур і критеріїв приймання результатів.

І хоча даний етап розробки проекту є надзвичайно важливим, якість його виконання залишається достатньо низькою. Про це свідчить те, що 60% відхилених замовником програмних проектів не відповідали вимогам [16]. Однією з основних причин цього є недостатнє використання формальних методів для представлення

вимог, що ускладнює їх комунікацію на стадіях ЖЦ, а також трансформацію проміжних представлень при внесенні змін у вихідні вимоги [15] та впровадження засобів автоматизації відповідних процесів. Важливість використання формалізованих методів підтверджується ще й тим, що біля 30% всіх помилок у вихідних вимогах виявлялись на етапі їх формалізації [16].

Оскільки початок етапу розробки вимог виконується розробником спільно із користувачем (замовником), то представлення вимог є слабоформалізованим, а процедура розробки – досить трудомісткою. На практиці для представлення вимог до ПЗ використовують псевдомови і нотації у вигляді різного роду діаграм (UML діаграм, діаграм стану, потоків даних та ін.). Через це виникає необхідність застосування певного системного підходу для забезпечення узгодженості використання технологій, формулювання та адекватного відображення потреб замовника в ПЗ на формалізовані специфікації вимог до ПЗ. Крім того, використовуванні формалізації вимог повинні забезпечити можливість їх трасування (комунікації) на наступних стадіях ЖЦ розробки ПЗ.

Провівши аналіз процесів, характерних стадії розробки вимог, можна зробити висновок, що основними факторами негативного впливу на якість ПЗ є:

- висока трудомісткість етапу формулювання потреб замовника;
- недосконаліть або відсутність методів та інструментальних засобів уніфікованого та формалізованого представлення вимог;
- складність забезпечення системного підходу розробки вимог, їх комунікації та відображення на наступних етапах ЖЦ.

Причиною більшості відхилень проектів є неправильне, нечітке, або незрозуміле формулювання вимог до ПЗ. Проблеми, що приводять до невдалого виконання проекту можна поділити на три категорії, дві з яких пов’язані з неякісно розробленими потребами:

- Вимоги. Погано організовані, погано написані, слабо зв’язані з потребами зацікавлених сторін, досить швидко змінюються, або змінюються безпідставно, не відповідають реальності.

• Недостатність ресурсів. Недостатність фінансування, дисципліни при проектуванні, погана організація вимог та управління вимогами.

Основні фактори негативного впливу на якість виконання проекту у вигляді відсоткового еквіваленту наведено на рис. 1.2.

Рис. 1.2 – Фактори, що негативно впливають на результат розробки

Як видно з рис. 1.2 забезпечення належної якості розробки вимог є визначальною, оскільки зменшує кількість відхилених проектів на 35%.

Наступним важливим процесом є проектування і (або) вибір архітектури ПЗ. Так у [15] визначено декілька критеріїв, які характеризують архітектуру в аспекті структурної залежності модулів та зв’язків всередині них. Однак невеликий набір цих критеріїв недостатній для повноти опису та вибору оптимальної архітектури. Крім того, така технологія вибору достеменно не відображає бізнес-логіку ПЗ. Тому для побудови або вибору оптимальної архітектури необхідно забезпечити відображення користувацьких вимог якості на повну множину критеріїв до ПЗ на етапі проектування архітектури. Для цього необхідно розробити процедури формалізованого перетворення та комунікації вимог якості на усіх етапах ЖЦ. У зв’язку з їх відсутністю, сучасні технології не забезпечують виконання первинних

вимог, сформульованих замовником, при проектуванні архітектури, що може привести до неможливості задоволення цих вимог у кінцевому програмному продукті.

Іншим підходом до проектування архітектур є спосіб розподілу функцій ПЗ за шарами. Однак формалізованої процедури для відображення користувацьких вимог на вимоги компонентів, або відповідних функцій за шарами не розроблено. Крім того, при використанні такого підходу, складність розподілу функцій за шарами є досить високою. На практиці ці технології застосовуються фірмами-розробниками для проведення декомпозиції ПЗ, але при цьому не розглядаються задачі забезпечення ефективності, оптимальності та якості процесу проектування. З проведеного вище аналізу випливає, що проблеми забезпечення якості ПЗ при проектуванні архітектури в основному пов’язані з відсутністю формалізованих процедур комунікації вимог якості на даний етап, а також використанням не повного набору критеріїв вибору або оцінювання якості характеристик існуючих типів архітектур.

Провівши аналіз стадій ЖЦ ПЗ з точки зору забезпечення якості їх виконання, можна зробити висновок про те, що на кожному з них існує ряд проблем, пов’язаних з методами і технологіями забезпечення якості.

В першу чергу це пов’язано із застосуванням слабоформалізованих процедур відображення вимог до ПЗ, як на етапі їх розробки так і на наступних стадіях ЖЦ.

По-друге, критерії оцінювання якості, які використовують розробники на відповідних стадіях проектування, є корпоративними, не уніфікованими, не стандартизованими та не повними.

По-третє, процедури відстеження (комунікації) вимог не забезпечують адекватності і об’єктивності їх представлення на наступних стадіях ЖЦ, що пов’язано з відсутністю єдиного уніфікованого підходу щодо їх комунікації.

Виходячи з цього, забезпечення ефективності розробки ПЗ та управління якістю на усіх етапах ЖЦ відіграє досить важливу роль і є однією з основних задач інженерії програмного забезпечення. Для підвищення якості як технологічних

процесів окремих етапів, так і процесу розробки в цілому, потрібно впроваджувати більш ефективні методи забезпечення та контролю якості, спрямовані на адекватне та повне відображення потреб користувача (замовника). Для підвищення якості розробки ПЗ пропонується використати апарат моделей якості та відповідних методів їх побудови. Проведемо аналіз існуючих моделей якості та вимог до них з точки зору можливості їх використання на стадіях ЖЦ ПЗ для забезпечення належного рівня задоволення потреб користувачів.

1.2 Моделі якості ПЗ

Побудова та використання моделей якості обумовлена необхідністю структурованого представлення сукупності властивостей ПЗ при виконанні проекту і способів їх вимірювання. Виходячи з означення якості, як властивості ПЗ задовольняти потреби замовника (користувачів) (з одного боку і міри відповідності вимогам з іншого) модель якості можна трактувати наступним чином: це сукупність властивостей ПЗ та зв’язків між ними, які становлять базу для специфікації вимог та їх оцінювання. Як наслідок, очевидними вимогами до таких моделей є адекватність і повнота відображення вимог якості, наявність засобів оцінювання рівня їх забезпечення, здатність моделі до трансформації та адаптації на різних етапах розробки проекту. Оскільки, вхідними даними для розробки ПЗ є потреби замовника і результати аналізу предметного середовища, то для їх опису на основі моделей якості необхідно визначити сукупність атрибутів якості і встановити їм відповідні метрики.

Хоч розроблено та введено в дію стандарти з якості ПЗ, в яких міститься уніфікована термінологія і набір метрик [8,9,10], та все ж на практиці та у наукових публікаціях [16] продовжують використовувати не стандартизовану термінологію і не уніфіковані характеристики якості, що приводить до неможливості порівняння результатів досліджень та використання їх на практиці. Особливо різне семантичне

навантаження несе термін «метрика», стандартизоване представлення якої в системі виміру якості наведено на рис. 1.3.

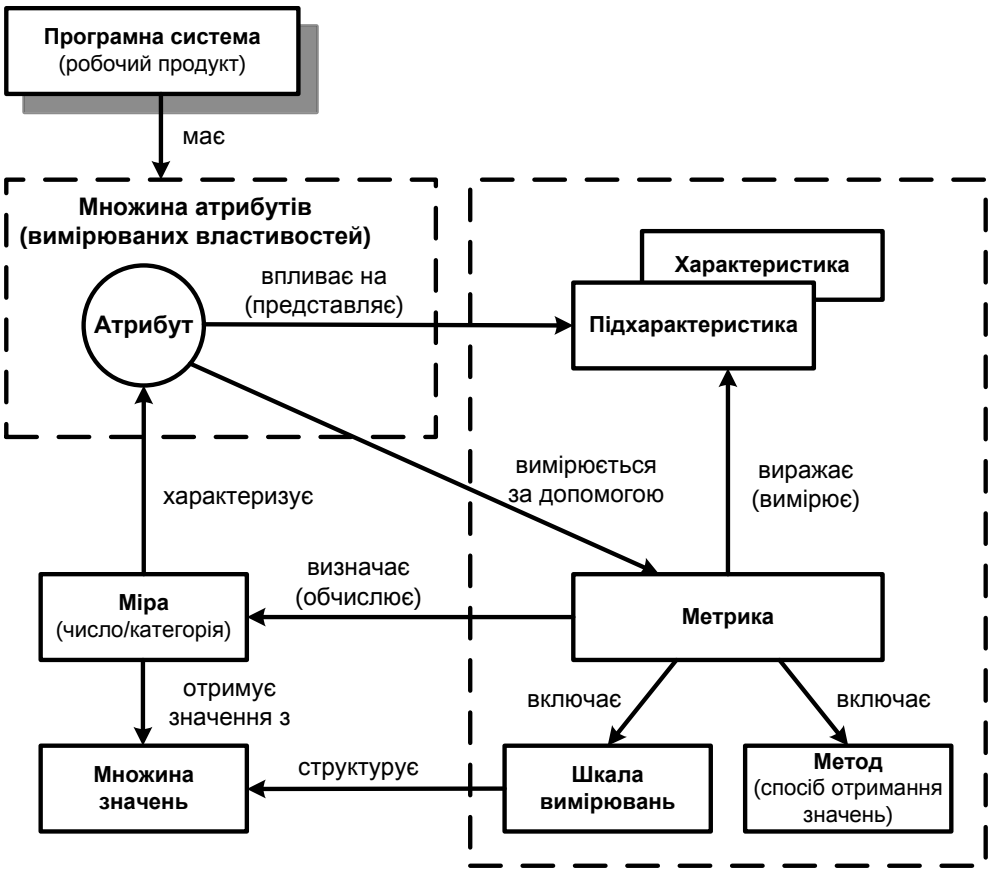

Рис. 1.3 – Метрика в системі виміру якості

Наведемо означення терміну «метрика» згідно стандарту [17]. Метрика – це комбінація конкретного методу виміру (способу одержання значень) атрибута сутності й шкали виміру (засобу, що використовується для структурування одержаних значень). Метрика визначає міру атрибута – змінну, якій привласнюється значення в результаті виміру [15].

На основі означення моделі якості та висунутих до неї вимог можна зробити висновок про те, що ефективність її використання на стадіях ЖЦ залежить від повноти набору характеристик якості, методів і способів кількісного їх оцінювання, наявності процедури трансформації моделі відповідно до етапу розробки.

Проведемо аналіз моделей якості, які можна використовувати при проектуванні та оцінюванні якості ПЗ.

Перша спроба систематизувати та структурувати характеристики якості програмних продуктів належить Дж. МакКолу [18]. Запропонована ним модель якості включає характеристики, які відображають властивості ПЗ з погляду користувача і розробника. У загальному випадку елементи моделі якості МакКола можна поділити на три групи. До першої групи належать характеристики якості програмного продукту, які сформульовано замовником та користувачем (factors), що фактично відображає їх потреби у ПЗ. Друга група елементів включає критерії якості (criteria), які описано з точки зору розробників. Ці критерії формують цілі проекту. До третьої групи характеристик якості можна віднести атрибути, означені автором як «метрики» (metrics).

Наведена на рис. 1.4 модель на той час була великим кроком вперед в інженерії якості ПЗ, оскільки дозволяла структурувати критерії оцінювання властивостей програмних продуктів. Методи встановлення ваги характеристик якості суб’єктивні, оскільки базуються на поглядах конкретних розробників. Недоліком застосування моделі МакКола є те, що кожна «метрика» впливає на оцінку багатьох факторів якості. Кількісне вираження якості ПЗ проводиться на основі лінійних методів без врахування кореляції між характеристиками.

Рис. 1.4 – Модель МакКола

Для нівелювання недоліків підходу [18] в 1978 р. Б. Боемом запропоновано модель якості, яка в основному була орієнтована на розробників ПЗ. Праці спрямовувались на стандартизацію характеристик якості ПЗ, систем їхніх мір і як наслідок сприяли розвитку досліджень у сфері контролю і керування якістю. Це стосується робіт, які базувались на використанні характеристик моделі Боема, зокрема надійності програмного забезпечення.

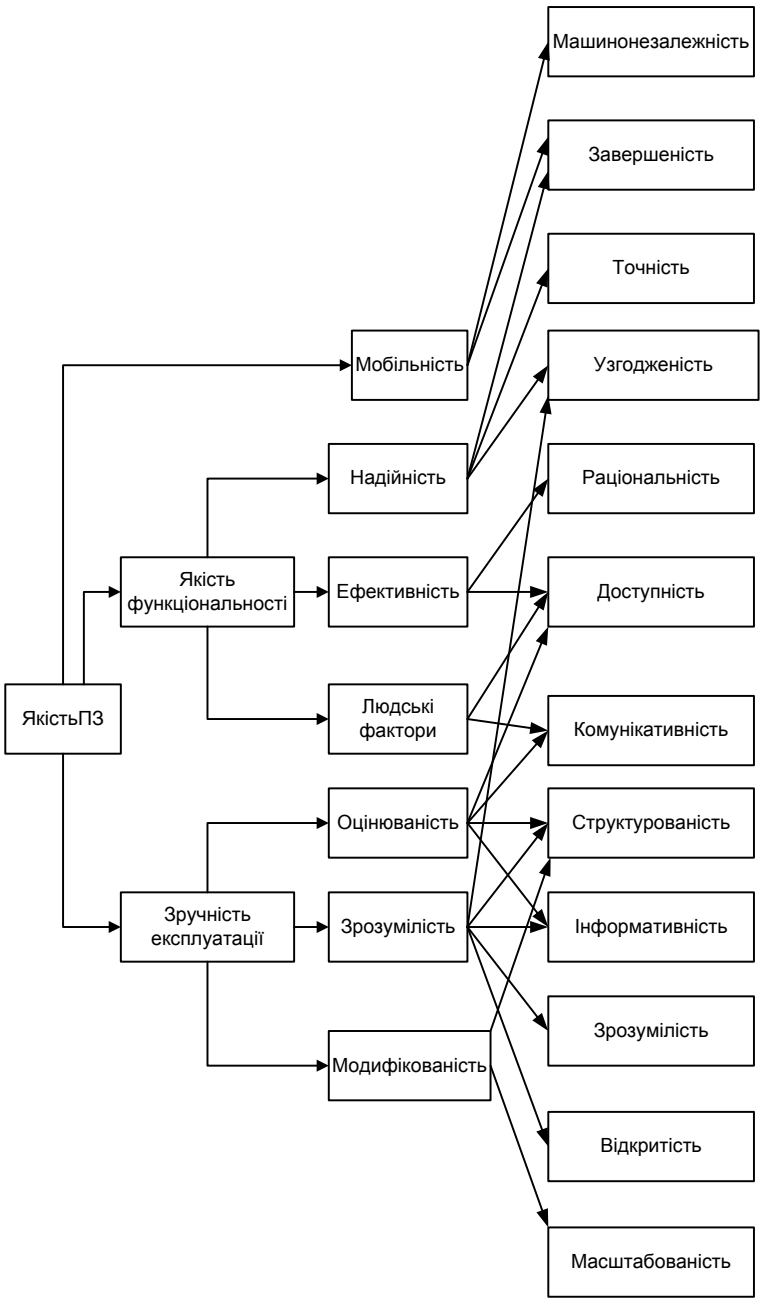

Рис. 1.5 – Модель Боема

Крім запропонованих у [18] властивостей якості, у моделі Боема з'явились деякі додаткові: функціональність, універсальність, зрозумілість та інші. Модель [18] є дворівневою, яка включає характеристики якості і відповідні їм атрибути, а модель [11] представляє собою трьохрівневе ієрархічне дерево, яке включає характеристики, що об'єднані у три групи, «атрибути», які фактично є підхарактеристиками, та метрики з допомогою яких можна проводити оцінювання властивостей програмного продукту. Нечітка структуризація з перекриттям зв'язків «характеристика-підхарактеристика» приводить до неможливості отримання оцінок характеристик за оціненими значеннями підхарактеристик. Модель Боема включає 19 атрибутів якості, які розширюють модель МакКола (рис. 1.5).

Застосування моделі [11] для забезпечення якості на різних стадіях розробки проекту практично неможливе, оскільки вона є розширенням моделі якості [18], недоліки якої визначено у попередньому пункті.

Софтверні компанії такі, як Motorola, Hewlett Packard, Canon, IBM використовують системи управління та контролю якості, які базуються на корпоративних моделях якості ПЗ. Оскільки, продукти цих фірм випускаються масштабними серіями та орієнтовані на значну аудиторію користувачів, то їх ключовими завданнями є задоволення користувацьких потреб.

Для досягнення цієї мети розробники серійного ПЗ використовують моделі якості FURPS (functionality, usability, reliability, performance, service) та CUPRIMDSO (capability, usability, performance, reliability, installability, maintainability, documentation/information, service, overall). Структура моделі FURPS наслідувана з моделей МакКола та Боема. Однак така характеристика якості як переносимість у цій корпоративній моделі, на відміну від попередніх, окремо не виділяється, а є складовою інших характеристик. При цьому здійснити комунікацію вимог якості та провести їх оцінювання досить складно, оскільки необхідно враховувати аспекти (пріоритетність атрибутів) різних характеристик моделі FURPS. Модель CUPRIMDSO є більш повною по відношенню до FURPS, оскільки має ширший

набір характеристик якості, що дозволяє точніше виявити рівень задоволення реалізованих у ПЗ властивостей.

Спільним для цих двох моделей є те, що вони дозволяють виражати якість ПЗ як з точки зору кінцевого користувача так і збоку самої ПЗ. Тому якість ПЗ, виходячи із застосування цих моделей, можна поділити на три категорії:

- якість продукту;
- внутрішня якість процесів;
- якість підтримки.

Моделі CUPRIMDSO та FURPS використовують при оцінюванні якості програмного продукту для встановлення міри його відповідності користувацьким потребам. Внутрішня якість процесів на етапах розробки проекту контролюється шляхом виявлення дефектів коду. Однак методів і засобів прямого відображення вимог якості на вимоги внутрішньої якості процесів на стадіях ЖЦ не існує. Це може призводити до недостовірного прогнозування якості нових версій продукту, і як наслідок підвищення рівня якості одних характеристик негативно відображається на інших.

Забезпечення необхідного рівня якості сучасних програмних комплексів можливе при використанні міжнародних стандартів, розроблених та затверджених при участі представників провідних компаній галузі. Так, дослідження характеристики надійність та продуктивність обумовлено тим, що для розрахунку їх фактичного значення ефективно застосовувались засоби тестування. Серед робіт, присвячених продуктивності та надійності ПЗ, варто виділити праці, які згодом були відображенні та систематизовані у рекомендаціях стандарту ISO 9126.

Якість ПЗ в [17] визначена як: «сукупність властивостей ПЗ, які виражають ступінь задоволення та відповідності потребам». При цьому у стандарті розрізняють три категорії якості:

- якість у використанні – сукупність властивостей ПЗ, що визначають міру досягнення користувачами поставлених цілей у визначеному середовищі та відповідному контексті експлуатації.

- зовнішня якість – сукупність властивостей ПЗ, які виражають ступінь відповідності вимогам при її використанні у середовищі розробника з набором тестових даних.
- внутрішня якість – сукупність властивостей ПЗ, які відображають ступінь задоволення внутрішніх вимог якості і може бути визначена шляхом проведення інспекцій коду, тестування та рев'ю.

У стандарті [17] для кожної категорії якості запропоновано відповідні моделі у вигляді набору характеристик і взаємозв'язків між ними, що формують базу для специфікації вимог і оцінювання якості.

Так, для представлення якості у використанні запропоновано 4 характеристики якості, а для зовнішньої та внутрішньої – 6. З метою деталізації характеристик зовнішньої та внутрішньої якості ПЗ передбачено відповідні набори підхарактеристик. Така структура моделей дозволяє адекватно та повній мірі провести оцінювання властивостей ПЗ відносно вимог до неї.

Взаємозв'язок між категоріями якості стандарту [17], показано на рис. 1.6.

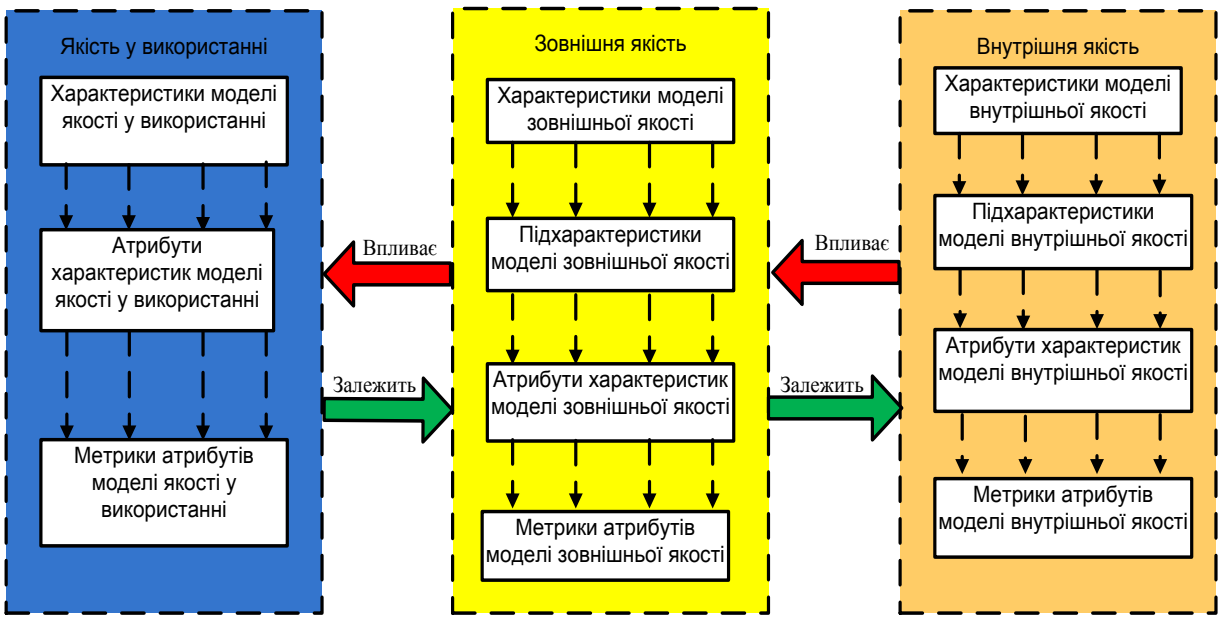

Рис.1.6 – Модель взаємозв'язку між категоріями якості ПЗ: якість у використанні (а); зовнішня якість (б); внутрішня якість (в)

Оскільки, згідно рекомендацій стандарту [7] визначено три моделі, призначених для оцінювання якості програмного продукту, то їх можна використати для формулювання та комунікації вимог якості на етапах ЖЦ.

Такий підхід дозволяє забезпечити якість виконання проекту на усіх фазах розробки. Крім того, на базі моделей в термінах стандартів [7, 8, 9, 10, 17] можна ефективно контролювати виконання процесів на кожному етапі ЖЦ і проводити керування ними.

1.3 Порівняльний аналіз моделей якості при їх застосуванні на стадіях ЖЦ ПЗ

У результати аналізу і порівняння була також включена і модель Дроми. Вона практично не відрізняється від моделі якості стандарту ISO 9126, який був прийнятий в 1991 р. Перевагою цієї моделі є те, що її використання дозволяє контролювати якість програмного продукту на етапах ЖЦ. Однак характеристики якості моделі Дроми є не уніфікованими, не стандартизованими і не набули широкого застосування.

Для проведення порівняльного аналізу моделей якості ПЗ визначено категорії критеріїв, які висуваються до такого класу моделей. Моделі якості повинні містити сукупність характеристик якості, набори метрик для проведення оцінювання реалізованих у ПЗ властивостей, забезпечувати здатність до адаптації на стадіях ЖЦ. Результати порівняльного аналізу моделей якості у відповідності до сукупності критеріїв до них наведено у табл. 1.1.

Виходячи з результатів наведених в табл. 1.1 можна зробити висновок, що найбільш ефективним при оцінюванні якості програмних продуктів, а також при забезпеченні якості процесів на стадіях ЖЦ ПЗ є застосування моделей якості стандарту ISO/IEC 25010.

Таблиця 1.1

Результати порівняльного аналізу моделей якості ПЗ

Вимоги до моделей	Модель МакКола	Модель Боема	Модель FURPS	Модель CUPRIMDSO	Модель Дроми	Моделі якості ISO 9126
Повнота визначення характеристик якості	+/-	+/-	+/-	+/-	+	+
Наявність метрик для кількісного вираження якості	-	+	+	+	+	+
Формалізованість та узгодженість із стандартами	-	-	-	+	-	+
Здатність до трансформації	-	-	+	-	+	+
Можливість комунікації вимог на стадіях ЖЦ	-	-	-	-	-	+
Ідентифікація вимог якості	-	-	+	+	+	+
Структурованість та адаптація до предметної області	+/-	+/-	+	+	+	+
Ідентифікація цілей проекту	-	-	+	+	+	+

Однак прямо застосувати їх на стадіях ЖЦ досить складно, тому необхідно розробити методику для системного та адекватного відображення характеристик якості та забезпечити гнучкість і модифікованість цієї моделі. А це в свою чергу потребує розробки формалізованого апарату представлення моделей якості і відповідних процедур їх трансформації та адаптації на кожному з етапів розробки.

Особливо важливим є застосування системного підходу на ранніх стадіях ЖЦ, оскільки від них найбільше залежить якість програмного продукту.

У результаті проведеного аналізу досліджень виявлено, що для забезпечення якості ПЗ розроблено велику кількість різних методів та засобів, орієнтованих на задоволення заявлених у специфікації вимог. Хоч методи ітераційного тестування на етапах розробки ПЗ застосовують на практиці, але цей підхід не дає можливості визначити міру задоволення нефункціональних вимог і обмежень. Це пов’язано із впливом суб’єктивних факторів на тестування інтерфейсів програмного продукту, відсутністю стандартизованих та уніфікованих метрик в процесі тестування, науково-обгрунтованої концепції проведення тестування на різних стадіях ЖЦ.

В першу чергу це пов’язано з тим, що вони статичні, і як наслідок застосовувати одну і ту ж модель на різних етапах розробки досить складно.

По-друге, набір характеристик якості не структурований, не завжди повно та адекватно відображає атрибути предметної області та не містить повного набору метрик для вираження кількісної міри задоволення потреб замовника.

По-третє, процедура комунікації вимог якості неформалізована і не дозволяє забезпечити їх комунікацію.

Для забезпечення якості процесу розробки ПЗ, зокрема на стадії розробки вимог та їх комунікації запропоновано використати моделі якості, які будуються виходячи із рекомендацій стандарту ISO 25010.

Виходячи з вище наведеного, можна зробити висновок, що основними задачами забезпечення якості програмного продукту в процесі його розробки є:

1. Створення формалізованої процедури побудови моделей якості в термінах стандарту [17] на стадіях ЖЦ.
2. Розробка методу перетворення потреб замовника у специфікацію вимог до ПЗ та комунікація їх на стадіях ЖЦ.
3. Забезпечення гнучкості, здатності до модифікації та повторного використання основних характеристик моделі якості [17].

4. Розробка автоматизованих інструментальних засобів підтримки процесів побудови моделі якості ПЗ та проведення процесу її оцінювання.

Тому наступний розділ наукового дослідження присвячено саме методам, процедурам і технології розробки та комунікації вимог якості на основі моделей якості.

1.4 Характеристика архітектури ПЗ

Архітектура програмного забезпечення є важливий аспект якісного програмного продукту який впливає на працездатність програми і її життєвий цикл.

Архітектура програмного забезпечення – це структура програми або обчислювальної системи, яка містить програмні компоненти, видимі зовні властивості цих компонентів, а також відношення між ними. Цей термін також відноситься до документування архітектури програмного забезпечення. Документування архітектури програмного забезпечення спрощує процес комунікації між зацікавленими особами, дозволяє зафіксувати прийняті на ранніх етапах проектування рішення про високорівневий дизайн системи і дозволяє використовувати компоненти цього дизайну і шаблони повторно в інших проектах.

Область комп'ютерних наук з моменту свого утворення зіткнулася з проблемами, пов'язаними зі складністю програмного забезпечення. Раніше проблеми складності вирішувалися розробниками шляхом правильного вибору структур даних, розробки алгоритмів і розмежуванню повноважень. Хоча термін «архітектура програмного забезпечення» є відносно новим для індустрії розробки програмного забезпечення, фундаментальні принципи цієї області невпорядковано застосовувалися починаючи з середини 1980-х. Перші спроби зрозуміти і пояснити програмну архітектуру системи були повні неточностей і страждали від нестачі організованості, часто це була просто діаграма з блоків, з'єднаних лініями. В 1990-ті роки спостерігається спроба визначити і систематизувати основні аспекти даної дисципліни. Початковий набір шаблонів проектування, стилів дизайну, передового

досвіду, мови опису та формальна логіка були розроблені протягом цього часу. На сьогоднішній день до цих пір немає згоди щодо чіткого визначення терміну «архітектура програмного забезпечення».

Будучи, наразі, дисципліною без чітких правил про «правильний» шлях створення системи, проектування архітектури програмного забезпечення є поєднанням науки і мистецтва. Аспект мистецтва полягає в тому, що будь-яка комерційна система передбачає наявність застосування або місії. Ключові цілі, які має система, описуються з допомогою сценаріїв як не функціональні вимоги до системи, також відомі як атрибути якості, які визначають, як буде вести себе система. Атрибути якості системи включають в себе відмовостійкість, збереження зворотної сумісності, розширюваність, надійність, придатність до сервісного обслуговування, доступність, безпека, зручність використання, а також інші якості. З точки зору користувача програмної архітектури, програмна архітектура дає напрям руху і вирішення завдань, пов'язаних зі спеціальністю кожного такого користувача, наприклад, зацікавленої особи, розробника ПЗ, групи підтримки, спеціаліста по супроводу, спеціаліста по розгортанню ПЗ, тестера, а також кінцевих користувачів. У цьому сенсі архітектура програмного забезпечення насправді об'єднує різні точки зору на систему. Той факт, що ці кілька різних точок зору можуть бути об'єднані в архітектурі програмного забезпечення є аргументом на захист необхідності і доцільності створення архітектури ще до етапу розробки ПЗ.

Початок архітектурі програмного забезпечення як концепції було покладено в науково-дослідницькій роботі Едсгера Дейкстри в 1968 році і Девіда Парнаса на початку 1970-х. Ці вчені підкреслили, що структура системи ПЗ має важливе значення, і що побудова правильної структури - критично важливо. Популярність вивчення цієї області зросла з початку 1990-х років разом з науково-дослідницькою роботою по дослідженню архітектурних стилів (шаблонів), мов описання архітектури, документування архітектури і формальних методів.

У розвитку архітектури програмного забезпечення як дисципліни відіграють важливу роль науково-дослідницькі установи. Мері Шоу і Девід Гарлан з

університету Carnegie Mellon написали книгу під назвою «Архітектура програмного забезпечення: перспективи нової дисципліни в 1996 році», в якій висунули концепції архітектури програмного забезпечення, такі як компоненти, з'єднувачі, стилі. Першим стандартом програмної архітектури є стандарт IEEE 1471: ANSI / IEEE 1471–2000: Рекомендації по опису переважно програмного забезпечення. Він був прийнятий в 2007 році, під назвою ISO ISO / IEC 42010:2007.

1.5 Порівняння методів проектування архітектури ПЗ

Проектування архітектури ПЗ – це процес розроблення, що виконується після етапу аналізу і формування вимог. Задача такого проектування – перетворення вимог до системи у вимоги до ПЗ і побудова на їхній основі архітектури системи [20]. Побудова архітектури системи здійснюється шляхом визначення цілей системи, її вхідних і вихідних даних, декомпозиції системи на підсистеми, компоненти або модулі та розроблення її загальної структури. Проектування архітектури системи може проводитися різними методами (стандартизованим, об’єктно-орієнтованим, компонентним і ін.), кожний з яких пропонує свій шлях побудови архітектури, а саме, визначення концептуальної, об’єктної й інших моделей за допомогою відповідних конструктивних елементів (блок-схем, графів, структурних діаграм тощо). Розрізняють 3 основні види проектування архітектури програмного забезпечення.

Стандартизований підхід, тривалий час в Україні та СРСР розроблення програмного забезпечення виконувалась на основі стандарту [21], що регламентує стадії й етапи процесу розробки програмного забезпечення.

Об’єктний підхід, проектування системи може здійснюватися на основі об'єкто-орієнтованого моделювання методом UML, який дозволяє враховувати аспекти, властиві діючим особам (акторам) системи, створювати сценарії виконання системи. Об’єктний стиль проектування – це декомпозиція майбутньої системи на окремі підсистеми (пакети), визначення функціональних і не функціональних вимог

і об’єктної моделі предметної області. Носіями інтересів, можливостей і дій в системі (або пакеті) є діючі особи – актори. Пакет може складатися з об’єктної моделі, варіантів використання, що визначають сценарії поведінки системи, склад об’єктів і методів їхньої взаємодії. Поведінка об’єктів відображується діаграмами, що задають послідовність взаємодії об’єктів (діаграми послідовностей, взаємодії), правилами переходу від стану до стану (діаграми станів), а також діаграмами кооперації, в яких діючі особи зображуються графічно. Об’єкти і відповідні їм діаграми варіантів використання задають загальну архітектурну схему системи, у рамках якої здійснюється реалізація структури і специфіки поведінки компонентів. Загальна концепція об’єктного проектування – це побудова всіх сценаріїв, екранних діаграм для керування ними і їх випробування в різних варіантах використання. На вибір варіантів використання впливають не функціональні. На основі моделі опису вимог і понять проводиться уточнення складу і змісту функцій системи, методів їхньої реалізації у вигляді сценаріїв і діаграм потоків даних, у яких відображається взаємодія об’єктів як обмін повідомленнями між елементами системи для передачі даних і одержання відповіді після виконання операцій. Моделі вимог визначають призначення і місце вимог у таких системах. Цьому сприяють розроблені національні, корпоративні і відомчі стандарти. Вони фіксують правила формування не функціональних вимог, у яких відображаються відомості про взаємодію і захист даних у системі. При цьому поведінка об’єктів представляється діаграмами UML, вони можуть уточнюватися при перегляді моделей вимог і складу об’єктів системи. Перегляд починається з вимог і пошуку місць локалізації для внесення необхідних змін у модель. Зміни можуть стосуватися функціональних і не функціональних вимог у зв’язку з уточненням замовником обмежень на структуру системи, використовувані ресурси й умови середовища її функціонування.

Загальносистемний підхід, один зі шляхів архітектурного проектування – традиційний неформальний підхід до визначення архітектури системи, її компонентів, способів їхнього подання й об’єднання в систему, який можна назвати

загальносистемним. Фактично архітектура, що створюється згідно з таким підходом, є чотирирівневою і містить у собі:

Перший рівень – системні компоненти. Вони здійснюють взаємодію з периферійними пристроями комп'ютерів (принтери, клавіатура, сканери, маніпулятори і т.п.), використовуються при побудові операційних систем.

Другий рівень – загальносистемні компоненти. Вони забезпечують взаємодію з універсальними сервісними системами середовища роботи прикладної системи, такими як операційні системи, СКБД, системи баз знань, системи керування мережами.

Третій рівень – специфічні компоненти певної прикладної області, що входять до складу компонентів програмної системи і призначені для розв'язання задач в межах означеної області (наприклад, бізнес-задачі).

Четвертий рівень – прикладні програмні системи, що призначені для виконання завдань з обробки інформації, які постають перед окремими групами споживачів інформації з різних предметних областей (офісні системи, системи бухгалтерського обліку й ін.) і можуть використовувати компоненти нижчих рівнів. Компоненти кожного з виділених рівнів використовуються, як правило, на своєму або вищому рівні. Кожен рівень відбиває відповідний набір знань, умінь і навичок фахівців, що створюють або використовують компоненти. Цей набір визначає відповідний розподіл фахівців програмної інженерії на аналітиків, системщиків, прикладників, програмістів й ін. При проектуванні архітектури програмна система розглядається як композиція компонентів третього рівня з доступом до компонентів першого і другого рівнів. Тобто архітектурне проектування – це розроблення компонентів третього рівня, визначення вхідних і вихідних даних рівнів ієрархії компонентів і їхніх зв'язків. Результат проектування – архітектура й інфраструктура, що містять у собі набір об'єктів, з яких можна формувати деякий конкретний вид архітектурної схеми для конкретного середовища виконання системи, а також набір елементів керування і контролю. Проектування архітектури системи завершується

створенням опису, в якому відображені зафіксовані проектні рішення, логічна і фізична структура системи, а також способи взаємодії об’єктів.

Новим методом проектування архітектури програмного забезпечення є шаблони. Архітектурні шаблони програмного забезпечення – це шаблони програмного забезпечення, що являють собою звіт практик вирішення архітектурних проблем розробки програмного забезпечення. Архітектурні шаблони виражають фундаментальну схему структурної організації певної програмної системи. Така схема складається із визначених наперед підсистем, а також точно визначає їхні сфери відповідальності та взаємовідносини.

Шаблони проектування програмного забезпечення – ефективні способи вирішення задач проектування програмного забезпечення. Шаблон не є закінченим зразком, який можна безпосередньо транслювати в програмний код. Об'єктно-орієнтований шаблон найчастіше є зразком вирішення проблеми і відображає відношення між класами та об'єктами, без вказівки на те, як буде зрештою реалізоване це відношення.

Головна користь кожного окремого шаблону полягає в тому, що він описує рішення цілого класу абстрактних проблем. Також той факт, що кожен шаблон має своє ім'я, полегшує дискусію про абстрактні структури даних (ADT) між розробниками, так як вони можуть посилатися на відомі шаблони. Таким чином, за рахунок шаблонів проводиться уніфікація термінології, назв модулів і елементів проекту. Правильно сформульоване шаблон проектування дозволяє, відшукавши вдале рішення, користуватися ним знову і знову.

1.6 Висновки до розділу

Основні результати, які отримані в даному розділі полягають в наступному:

1. Визначено місце та роль процесів забезпечення якості у загальній технології розробки ПЗ. Аналіз сучасного стану проблеми та наукових публікацій в цій області показав, що не дивлячись на важливість проблеми, на практиці ще

широко використовуються підходи, які базуються на тестуванні готового програмного продукту. І хоча, стандарти з якості були введені ще 10-15 років тому, однак їх широке впровадження на практиці проектування ПЗ стримується відсутністю методологічних основ та інструментальних засобів.

2. Проаналізовано сучасні підходи до забезпечення і контролю якості ПЗ та впровадження відповідних систем, які їх підтримують на стадіях ЖЦ ПЗ. Визначено, що технологія базована на застосуванні моделей якості є найбільш ефективною. Аналіз розроблених та використовуваних на практиці моделей якості показав, що найбільш повною, конструктивною і технологічною є модель якості побудована на основі рекомендацій стандарту ISO/IEC 25010.

3. Проаналізовано методи проектування архітектури ПЗ і обгрунтовано необхідність розробки методу та інструментального засобу оцінювання якості шаблонів проектування, що в подальшому дало б змогу забезпечити оптимальний вибір шаблонів архітектури.

РОЗДІЛ 2
МЕТОДИ АНАЛІЗУ ВИМОГ ТА ОЦІНЮВАННЯ ЯКОСТІ ПАТЕРНІВ АРХІТЕКТУРИ ПЗ

2.1 Дослідження і адаптація моделей якості стандарту ISO 9126 для представлення вимог до архітектури ПЗ

Відповідно до стандарту [7] модель якості ПЗ складається із трьох частин:

- внутрішня якість;
- зовнішня якість;
- якість у використанні.

Для адекватного застосування та адаптації відповідних моделей для представлення вимог до архітектури ПЗ, приведемо більш детальну схему залежностей між категоріями якості через їх характеристики (рис 2.1).

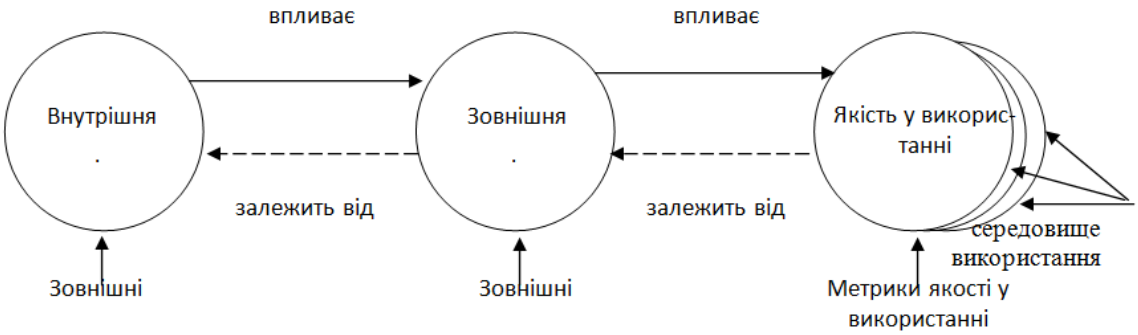

Рис. 2.1 – Залежності між категоріями якості стандарту ISO 9126

Виходячи із представлення залежності між категоріями якості (рис. 2.1), зміна характеристик однієї з них відображається відповідним чином на характеристиках всіх інших категорій. Тому підвищення внутрішньої якості впливає на підвищення якості всього ПЗ. Вимоги до архітектура програмного забезпечення визначать внутрішня якість ПЗ.

Для побудови внутрішньої моделі якості для архітектури програмного забезпечення, запропоновано використання моделі внутрішньої якості, що складається з характеристик, підхарактеристик, атрибутів та метрик. Атрибути, якими вимірюється рівень задоволення потреби по певній підхарактеристиці, вибираються, виходячи з аналізу предметного середовища використання ПЗ.

Вимоги внутрішньої якості ПЗ формулюються в термінах моделі внутрішньої якості. При цьому вимоги внутрішні якості повинні бути встановлені у специфікації на основі внутрішніх атрибутів. Структуру моделі зовнішньої якості у термінах характеристик і підхарактеристик наведено на рис. 2.2.

Рисунку 2.2 – Модель внутрішньої якості ПС

Вимоги внутрішньої якості використовують для визначення властивостей проміжних станів системи. Вимоги внутрішньої якості можуть бути використані для визначення стратегії подальшої розробки та можуть виступати в якості критеріїв оцінювання і верифікації процесу розробки. Формалізація вимог на етапі їх розробки забезпечує точність та простоту подальшої розробки проекту, зокрема це стосується вибору та побудови майбутньої архітектури ПЗ. Фактично модель внутрішньої якості, відображає та доповнює модель зовнішньої якості з погляду структури та функціональності модулів системи. Типи критеріїв для оцінювання характеристик і підхарактеристик внутрішньої моделі якості, наведено на рис. 2.2.

Для формалізації процедури побудови і представлення моделі внутрішньої якості запишемо її у вигляді множини:

$$\left\{H_i^x, \Pi_{ij}^x, M_{ij}^x\right\},\ i=\overline{1,6},\ j=\overline{1,K_i}, \tag{2.1}$$

де H_i^x – характеристики внутрішньої якості;

Π_{ij}^x – підхарактеристики внутрішньої якості;

M_{ij}^x – відповідні метрики;

K_i – кількість підхарактеристик і-ої характеристики.

Оскільки, модель внутрішньої якості повинна відображати потреби користувача на рівні проектування програмного забезпечення в цілому та його підсистем, то необхідні додати елементи множини, які не використовувались на попередньому етапі. В результаті отримаємо специфікації вимог зовнішньої якості R_{ext} в стандартизованих термінах.

$$R_{in}=\left\{H_i^x, \Pi_{iK}^x, A_{iK}^x, C_{iK}^x, M_{iK}^x\right\},\ i\in N_x,\ K=\overline{1,F_i^x}, \tag{2.2}$$

де $\{A_{iK}\}$ – множина атрибутів;

$C_{ік}$ - обмеження на потреби користувача.

Формування специфікацій вимог R_{in} до модулів програмного комплексу виконується відображення вимог зовнішньої якості на елементи структури моделі внутрішньої якості, в результаті отримується модель (2.2). Це відображення виконується для кожного модуля з врахуванням його функціональності або шляхом відображення на архітектурні шари. Метрики для (2.2) на основі асоціативних правил або застосування методу Text Mining визначено з відповідного розділу стандарту ISO 9126.

2.2 Розробка методу детектування архітектурних патернів ПЗ та оцінювання їх якості

Провівши аналіз процесу розробки ПС, існуючих підходів до проектування та аналізу вимог, на основі даних про предметну область побудовано концептуальну схему процесу детектування шаблонів проектування (рис. 2.3).

Процес детектування шаблонів базується на автоматичному аналізі моделей вимог та моделей шаблонів проектування. Моделі вимог та шаблонів можуть формуватися експертом виходячи із цілей розробки, специфіки процесу розробки чи прийнятих погоджень. Наприклад, модель вимог може базуватись на рекомендаціях стандартів [7,8,9,10] та базуватись на внутрішній моделі якості, а модель шаблонів – використовувати класифікацію подану у [25].

Оскільки існуючі стандарти та підходи до формування моделей вимог носять рекомендуючий та загальний характер, отримані моделі потребують адаптації для використання в конкретних умовах процесу розробки ПС, що визначаються предметною областю, типом ПС, цілями розробки.

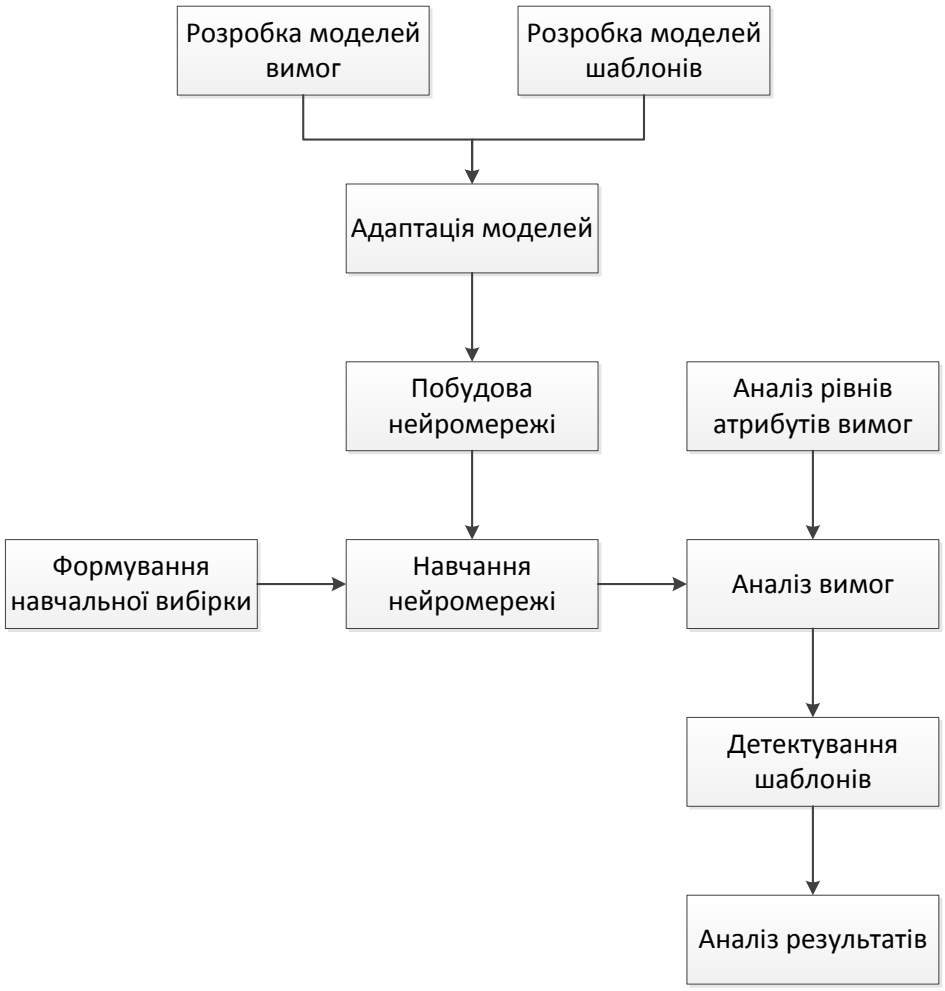

Рис. 2.3 – Концептуальна схема процесу детектування шаблонів проектування

Модель вимоги є сукупністю атрибутів, що формують множину $R\{A_1, \dots, A_n\}$, де значення атрибута визначається згідно шкали метрики для заданого атрибута. Шкала для значень атрибута може бути будь-якого типу, але повинна існувати її проекція на інтервальну шкалу $S_{Ai} \rightarrow S_{int}\{S_1,\dots, S_k\}$, $S_i \in [0;1]$. Сукупність вимог до ПС формує матрицю M

$$M = \begin{bmatrix} A_{11} & \dots & A_{1n} \\ \dots & \dots & \dots \\ A_{m1} & \dots & A_{mn} \end{bmatrix}, \tag{2.3}$$

де $A_{ij} \in S_{Aij}$ – атрибут вимоги R_i.

Модель шаблону проектування відповідно є сукупністю атрибутів, що формують множину P $\{K_1,\ldots,K_q\}$. Відносно шкали значень метрики атрибутів моделі шаблону проектування висувається аналогічна вимога існування проекції на інтервальну шкалу S_{int}: $S_{Ki} \rightarrow S_{int}$. Сукупність моделей шаблонів формують матрицю Q

$$Q = \begin{bmatrix} K_{11} & \ldots & K_{1q} \\ \ldots & \ldots & \ldots \\ K_{tq} & \ldots & K_{tq} \end{bmatrix}, \quad (2.4)$$

де $K_{ij} \in S_{Kij}$ – атрибут шаблону проектування Q_i.

Після адаптації моделей для конкретних вимог процесу розробки ПС формується відображення елементів матриць M та Q на шкалу S_{int}

$$M_{int} = \begin{bmatrix} I_{11} & \ldots & I_{1n} \\ \ldots & \ldots & \ldots \\ I_{m1} & \ldots & I_{mn} \end{bmatrix}, \quad (2.5)$$

де I_{ij} – відображення атрибуту A_{ij} на шкалу S_{int}

$$Q_{int} = \begin{bmatrix} Z_{11} & \ldots & Z_{1q} \\ \ldots & \ldots & \ldots \\ Z_{tq} & \ldots & Z_{tq} \end{bmatrix}, \quad (2.5)$$

де Z_{ij} – відображення атрибуту K_{ij} на шкалу S_{int}.

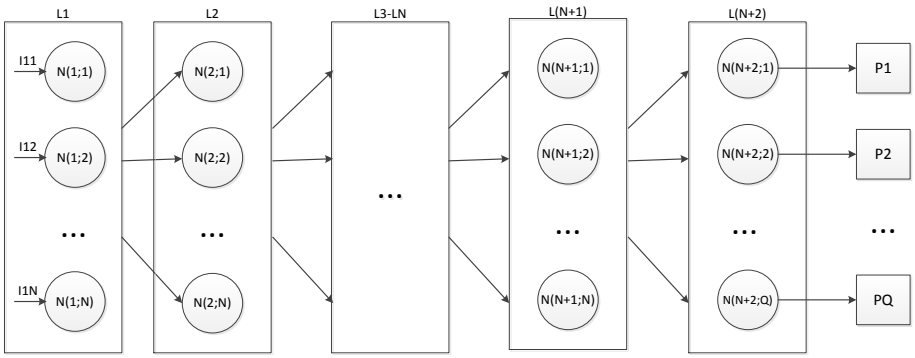

Рис. 2.4 – Будова нейромережі аналізатора моделі вимог

Основним компонентом процесу детектування шаблонів проектування є багаторівнева нейромережа (рис. 2.4). Розмірність нейромережі визначається розмірністю матриць M_{int} та Q_{int}. Вхідний рівень L1 містить штучні нейрони із лінійною перетворюючою функцією, що запобігає виродженню вхідного сигналу. На вхід нейронів рівня L1 подаються значення із рядків матриці M_{int}. Шар нейронів L2 містить штучні нейрони із сигмоподібною перетворюючою функцією та є першим рівнем прихованого шару нейромережі. Нейрони рівнів L3-LN є штучними нейронами із сигмоподібною перетворюючою функцією. Рівень L(N+1) складений із штучних нейронів із сигмоподібною перетворюючою функцією та є останнім рівнем прихованого шару. Рівень L(N+2) складений із штучних нейронів із сигмоподібною перетворюючою функцією та є вихідним рівнем нейромережі. Значення сигналів на виході шару L(N+2) формують ймовірні значення атрибутів моделі шаблонів проектування P (2.6).

$$\{M_{int}Q_{int}: L_j\} \rightarrow \begin{cases} L_j = 0 - \text{переходу немає} \\ L_j > 0 - \text{можливий перехід} \end{cases}, \quad (2.6)$$

де L_j – результуючий вектор,

j – добуток розмірності матриць M_{int} та Q_{int}.

На основі аналізу значень результуючого вектора на виході нейромережі методом простого ковзного середнього формується імовірність входження шаблону P_i в множину вимог M (2.7).

$$N(P_i M_{\text{int}}) = \frac{\sum(L_j)}{m}, \quad (2.7)$$

де m – кількість вимог.

2.3 Дослідження графічних нотацій систем детектування архітектурних патернів ПЗ

Для аналізу та проектування системи була обрана графічна нотація UML.

UML (Unified Modeling Language) — уніфікована мова моделювання, використовується у парадигмі об'єктно-орієнтованого програмування. Є невід'ємною частиною уніфікованого процесу розробки програмного забезпечення. UML є мовою широкого профілю, це відкритий стандарт, що використовує графічні позначення для створення абстрактної моделі системи, називаної UML-моделлю. UML був створений для визначення, візуалізації, проектування й документування в основному програмних систем. UML може бути застосовано на всіх етапах життєвого циклу аналізу бізнес-систем і розробки прикладних програм. Різні види діаграм які підтримуються UML, і найбагатший набір можливостей представлення певних аспектів системи робить UML універсальним засобом опису як програмних, так і ділових систем. Діаграми дають можливість представити систему у такому вигляді, щоб її можна було легко перевести в програмний код.

Діаграми створюються за допомогою спеціалізованих CASE-засобів, наприклад Rational Rose чи Enterprise Architect. На основі технології UML будується єдина інформаційна модель системи. UML прекрасно зарекомендувала себе в багатьох успішних програмних проектах. Засоби автоматичної генерації кодів дозволяють перетворювати моделі мовою UML у вихідний код об’єктно-орієнтованих мов програмування, що ще більш прискорює процес розробки.

2.3.1 Діаграма прецедентів (USE CASE)

Візуальне моделювання в UML можна представити як певний процес порівневого спуску від найбільш загальної і абстрактної концептуальної моделі вихідної системи до логічної, а потім і до фізичної моделі відповідної програмної системи. Для досягнення цих цілей спочатку будується модель у формі так званої

діаграми варіантів використання (use case diagram), яка описує функціональне призначення системи або, іншими словами, те, що система буде робити в процесі свого функціонування. Практично усі CASE-засоби (програми автоматизації процесу аналізу і проектування) мають підтримку UML. Моделі розроблені в UML, дозволяють значно спростити процес кодування і направити зусилля програмістів безпосередньо на реалізацію системи. Діаграми підвищують супроводжуваність проекту і полегшують розробку документації. UML необхідний:

- керівникам проектів, які керують розподілом завдань і контролем за проектом;
- проектувальникам інформаційних систем які розробляють технічні завдання для програмістів;
- бізнес-аналітикам, які досліджують реальну систему і здійснюють інжиніринг і реінжиніринг бізнесу компанії;
- програмістам які реалізовують модулі інформаційної системи.

При модифікації системи об'єктний підхід дозволяє легко включати в систему нові об'єкти і виключати застарілі без істотної зміни її життєздатності, що забезпечує низьку зв’язаність. Використання побудованої моделі при модифікаціях системи дає можливість усунути небажані наслідки змін, оскільки вони не ламають структури системи, а тільки змінюють поведінку об'єктів.

Діаграма прецедентів є графом, що складається з множини акторів, прецедентів обмежених границею системи, асоціацій між акторами та прецедентами, відношень серед прецедентів, та відношень узагальнення між акторами. Діаграми прецедентів відображають елементи моделі варіантів використання. Суть діаграми полягає в наступному: проектована система представляється у вигляді безлічі сутностей чи акторів, взаємодіючих із системою за допомогою так званих варіантів використання. Варіант використання (use case) служить для опису сервісів, що система надає актору. Іншими словами, кожен варіант використання визначає деякий набір дій, чинений системою при діалозі з

актором. При цьому нічого не говориться про те, яким чином буде реалізована взаємодія акторів із системою.

У мові UML є кілька стандартних видів відношень між акторами і варіантами використання:

• асоціації (association relationship)

• включення (include relationship)

• розширення (extend relationship)

• узагальнення (generalization relationship)

При цьому загальні властивості варіантів використання можуть бути представлені трьома різними способами, а саме — за допомогою відношень включення, розширення і узагальнення.

З даною системою будуть взаємодіяти 2 актора, тобто актори – це адміністратор системи і користувач.

Тут описується діаграма прецедентів використання для адміністратора.

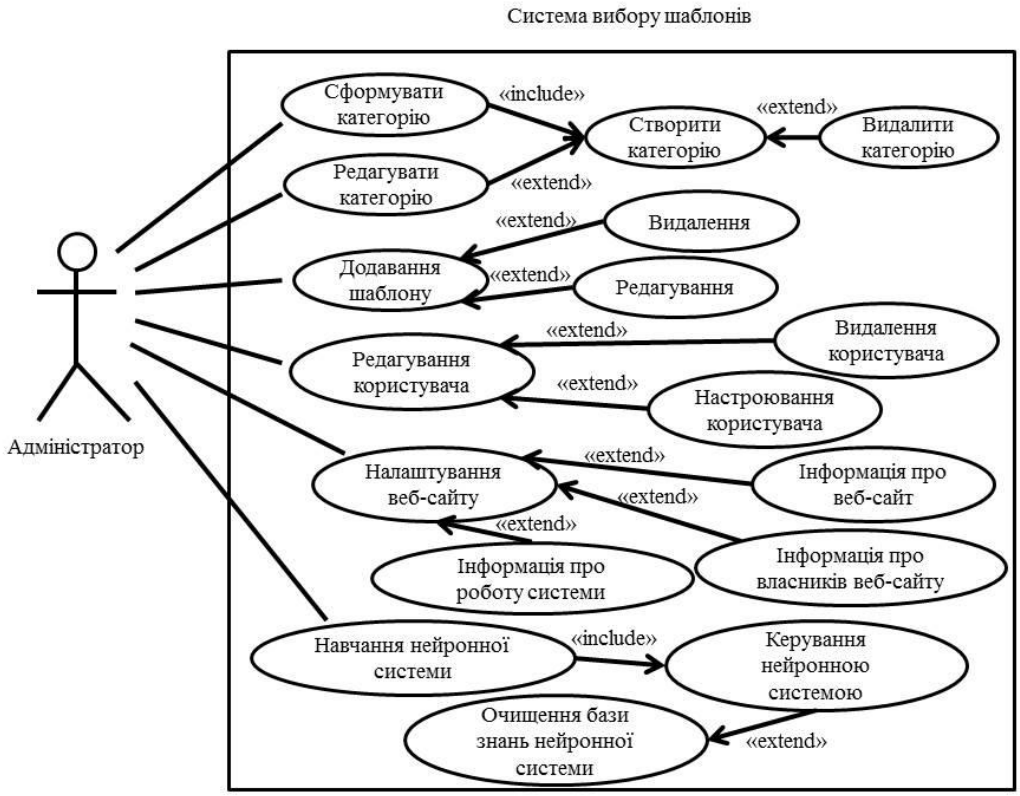

Рис. 2.1 – Діаграма прецедентів адміністратор

Сайт адміністрування призначений для виконання всіх поточних робіт з обслуговування веб-сайту. До нього мають доступ тільки співробітники та адміністрація веб-сайту.

Ось список основних завдань, що вирішуються сайтом адміністрування:

- формування і редагування структури категорій шаблонів;
- введення і редагування інформації про шаблони;
- прив'язка шаблону до категорії;
- перегляд, видалення, блокування і розблокування користувачів;
- отримання архівної та статистичної інформації шаблонів;
- навчання нейронної мережі або видалення її бази знань.

Структура категорій відображає категоризацію шаблонів, котрі розміщені на веб-сайті. Вони можуть змінюватися в міру зміни кількості патернів.

Каталог категорій має деревоподібну структуру, при цьому один і той же шаблон може розміщуватися в декількох рубриках цього каталогу, тобто може бути прив'язаний до різних гілок дерева.

У даному веб-сайті операції введення і редагування інформації про шаблони доводиться виконувати з появою нової інформації про них або при їх додаванні. Тому відповідні розділи сайту адміністрування повинні бути зручні у використанні.

При появі підозрілих дій користувача адміністратор може заблокувати його акаунт, якщо користувач звернеться з проханням розблокувати його до адміністрації або підозра про негативний вплив користувача на сайті зникне. Адміністратор може розблокувати його, в іншому випадку адміністратор може видалити користувача безповоротно.

Адміністратор також може редагувати сторінку зворотного зв’язку веб-сайту, щоб зробити інформацію на цій сторінці актуальною.

За допомогою інструментів адміністратор може навчати нейрону мережу, за допомогою навчальної вибірки або видаляти базу знань нейронної мережі.

На наступній діаграмі описується діаграма прецедентів використання для користувача.

З точки зору користувача, веб-сайт вибору архітектурних патернів складається з наступних основних компонентів:

- каталог шаблонів;
- системи регістрації, з вказанням контактної інформації;
- власний кабінет, де можна проглянути історію використання системи;
- зворотний зв'язок з адміністрацією сайту;
- система вибору патернів відповідно до вимого, та її подальше оцінювання;

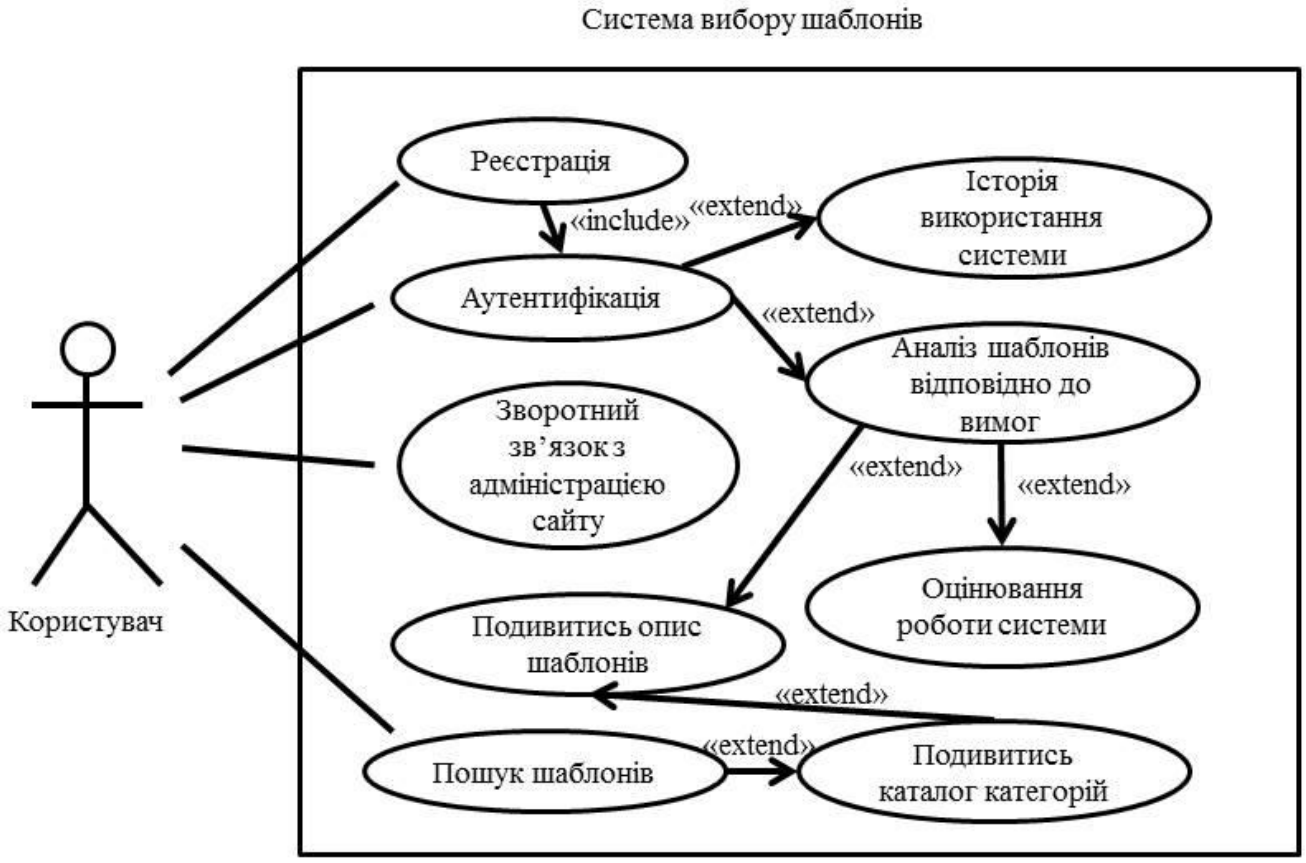

Рис. 2.2 – Діаграма прецедентів користувач

Каталог шаблонів являє собою частину бази даних веб-сайту, що зберігає всю інформацію про шаблони. Каталог має ієрархічну, деревовидну структуру, яка відображатиме спосіб класифікації шаблонів. Структура каталогу безпосередньо залежить від того, які шаблони є відомі на сайті.

При регістрації потрібно вказати контактну інформацію, а тобто потрібно вказати адресу електронної пошти, телефон, адресу, ім’я, пароль та інші дані.

Завершивши регістрацію користувач аутентифікується в системі і попадає в власний кабінет. З кабінету можна виконати пошук по шаблонів або виконати аналіз вибору шаблону відповідно до поставлених вимог. Також можна переглянути історію використання системи з її результатами.

Після аналізу вибору архітектурного шаблону і представлення результату користувач може оцінити роботу системи, вказавши оцінку роботи системи, добавити коментар до шаблону і добавити власну реалізацію шаблону.

Користувач може з легкістю відправити повідомлення адміністрації веб-сайту вказавши тему листа і текст листа, відповідно до того чи користувач аутентифікаваний потрібно вказати додатково як звертатись до користувача і електрону почту для отримання відповіді від адміністрації. Якщо він аутентифікований я то ця інформація береться з профіля користувача.

Також не аутентифікований користувач може ознайомитись з описом і реалізацією патернів, коментарями до них і статистику вибору.

Користувач може виконати пошук шаблонів по назві або тексті в описі шаблону скориставшись системою пошуку на головній сторінці.

2.3.2 Діаграма послідовності (Sequence Diagram)

Діаграма послідовності (Sequence Diagram) — діаграма, на якій зображена впорядкована в часі взаємодія об'єктів. Зокрема, на ній зображуються об'єкти, що беруть участь у взаємодії, і послідовність повідомлень, якими вони обмінюються.

Для моделювання взаємодії об'єктів у мові UML використовуються відповідні діаграми взаємодії. Взаємодію об'єктів можна розглядати в часі, і тоді для представлення тимчасових особливостей передачі і прийому повідомлень між об'єктами використовується діаграма послідовності. Об'єкти, що взаємодіють, обмінюються між собою деякою інформацією. При цьому інформація приймає

форму закінчених повідомлень. Іншими словами, хоча повідомлення і має інформаційний зміст, воно набуває додаткової властивості надавати напрямлений вплив на свого одержувача.

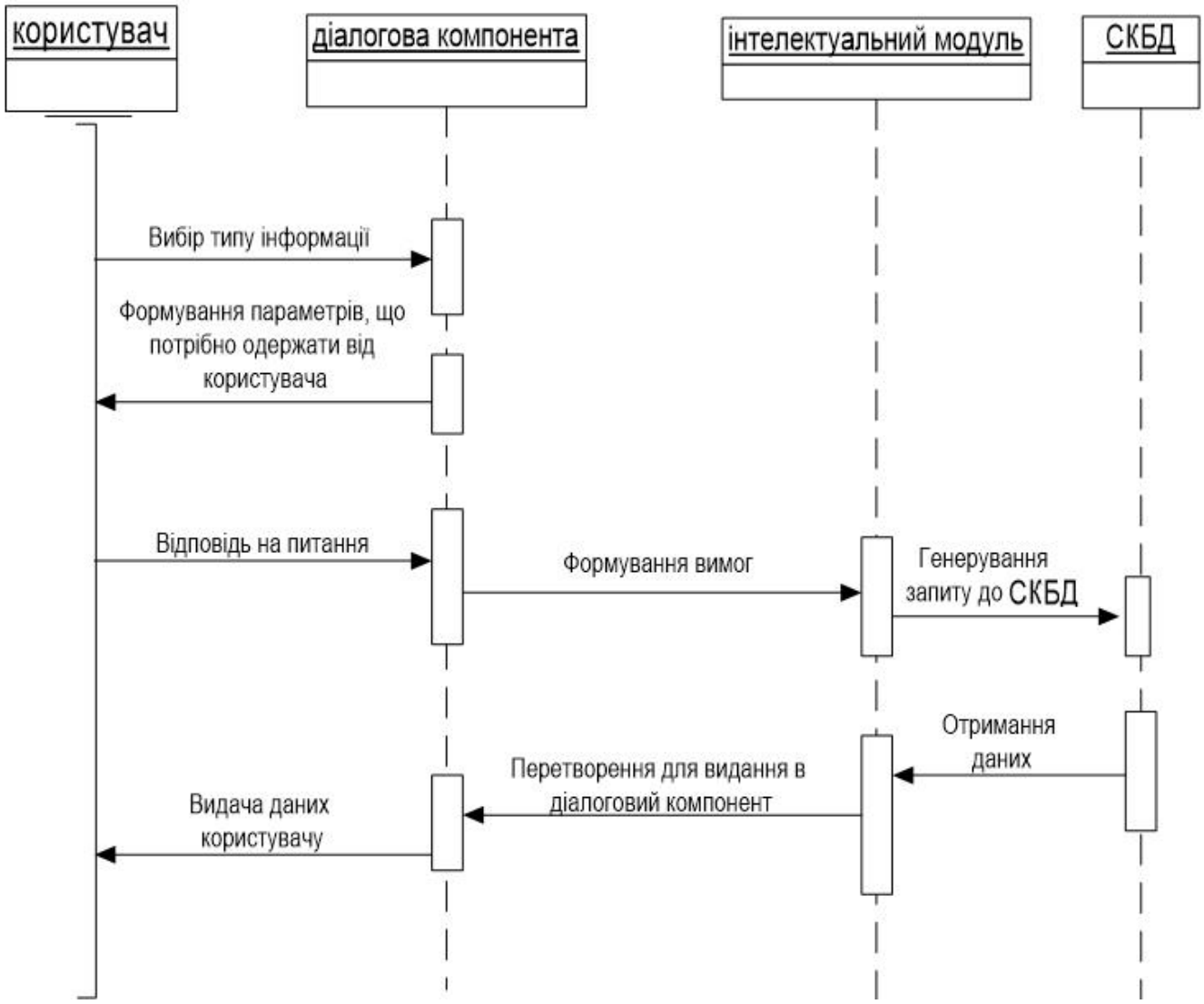

Рис. 2.3 – Діаграма послідовності

На цьому рис. 2.3. зображена діаграма послідовностей. Вона складається із 4 основних компонент: користувач, діалогова компонента, інтелектуальний модуль, СКБД.

- Користувач вибирає тип інформації і відповідає на запитання діалогової компоненти.

- У свою чергу діалогова компонента формує параметри, що потрібно отримати від користувача, видає дані адміністратору.
- Після цього діалогова компонента формує вимоги до інтелектуального модуля, який у свою чергу перетворює дані для видання у діалогову компоненту.
- Інтелектуальний модуль генерує запити в СКБД, а СКБД надсилає дані у інтелектуальний модуль, де формуються рекомендації на основі зібраних статистичних даних і передаються користувачу.

2.3.3 Діаграма класів (Class Diagram)

Діаграма класів — статичне представлення структури моделі. Відображає статичні елементи, такі як: класи, типи даних, їх зміст та відношення. Діаграма класів, також, може містити позначення для пакетів та може містити позначення для вкладених пакетів. Також, діаграма класів може містити позначення деяких елементів поведінки, однак їх динаміка розкривається в інших типах діаграм.

Діаграма класів показує, будівельні блоки будь-якої об’єктно-орієнтованої системи. Класові діаграми самі корисні в ілюстрації взаємозв’язків між класами і інтерфейсами. Узагальнення, з’єднання частин, і асоціації – уся цінність у віддзеркаленні спадкоємства, композиції або використання, і підключень відповідно.

В результаті робот виділимо такі класи (рис.2.4) : Користувач, Веб-сторінка, Веб-сервер, Інтелектуальна компонента, База даних та інтерфейс - Користувацький інтерфейс.

Користувацький інтерфейс зв’язує класи користувач та Веб-сторінка. Стрілками показана взаємодія цих класів між собою.

При об’єктно-орієнтованому підході до створення програмного забезпечення діаграма класів стає в нагоді, адже дозволяє вже на ранніх етапах побудови моделі також приблизно визначитись з архітектурою майбутнього програмного продукту.

Так, наприклад, клас Користувач має певний набір параметрів які його характеризують : ім’я,адресу електронної скриньки, права доступу тощо. Так само й інші класи.

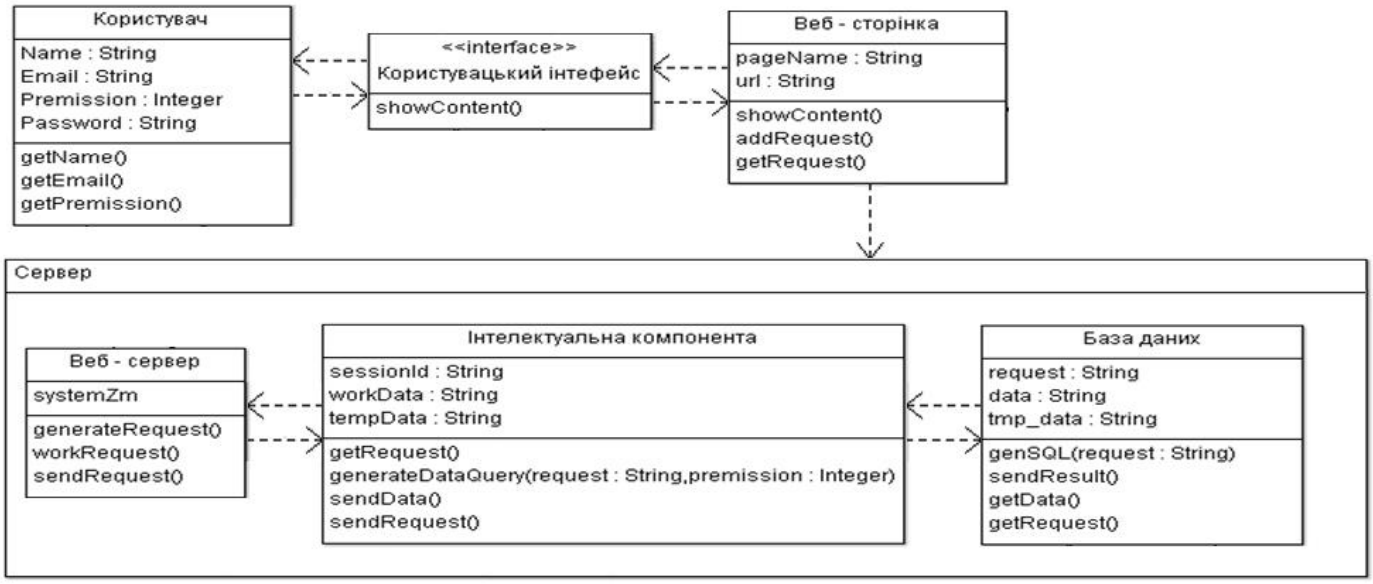

Рис. 2.4 – Діаграма класів

Варто звернути увагу на наявність інтерфейсу. Це вид класу який сам не має властивостей, але через нього відбувається взаємодія з іншими класами, в нашому випадку – класом Веб-сторінка.

Ще одним плюсом діаграми класів є те, що середовища розробки дозволяють генерувати програмний каркас на основі діаграми будь-якою мовою програмування (якщо вона підтримується середовищем).

2.3.4 Діаграма діяльності (Activity Diagram)

Діаграма діяльності — в UML, візуальне представлення графу діяльностей. Граф діяльностей є різновидом графу станів скінченного автомату, вершинами якого є певні дії, а переходи відбуваються по завершенню дій. Дія є фундаментальною одиницею визначення поведінки в специфікації. Дія отримує множину вхідних

сигналів, та перетворює їх на множину вихідних сигналів. Одна із цих множин, або обидві водночас, можуть бути порожніми. Виконання дії відповідає виконанню окремої дії. Подібно до цього, виконання діяльності є виконанням окремої діяльності, буквально, включно із виконанням тих дій, що містяться в діяльності. Кожна дія в діяльності може виконуватись один, два, або більше разів під час одного виконання діяльності. Щонайменше, дії мають отримувати дані, перетворювати їх та тестувати, деякі дії можуть вимагати певної послідовності. Специфікація діяльності (на вищих рівнях сумісності) може дозволяти виконання декількох (логічних) потоків, та існування механізмів синхронізації для гарантування виконання дій у правильному порядку. Варто зауважити що діаграма діяльності є статичним представлення структури звернень системи до БД в певний період часу.

Якщо користувач не авторизований,він має змогу це зробити, після чого він обирає як працювати з системою – переглядати інформаційне наповнення. Чи використати систему вибору шаблону відповідно до вимог. Оскільки після обрання категорії принцип роботи однаковий, формування запиту – вибірка з БД – виведення результату два варіанти з’єднуються вилкою для подальшого опрацювання. Дана діаграма показує які кроки виконуються перед зверненням системи до БД та після цього.

Для даної системи це виглядає наступним чином : Якщо користувач, авторизований чи ні, хоче працювати з системою, він спочатку має вибрати що саме збирається робити – задати критерії пошуку відразу, чи переглядати каталог категорій шаблонів. Відповідно до обраних дій до системи формується запит, інтелектуальна компонента обробляє його і передає іншій компоненті, яка формує запит до БД на основі отриманих даних. СКБД виконує запит та повертає його результат, ці дані знову форматуються для виведення та відображаються. Для авторизованих користувачів дозволяється використовувати систему вибору шаблонів відповідно до вимог, користувач задає критерії аналізу. Як і в попередньому випадку інтелектуальна компонента обробляє запит і передає іншій

компоненті яка з’єднується з БД і на основі оброблених даних з БД повертає результат користувачу.

Діаграма активності зображена на рисунку 2.5.

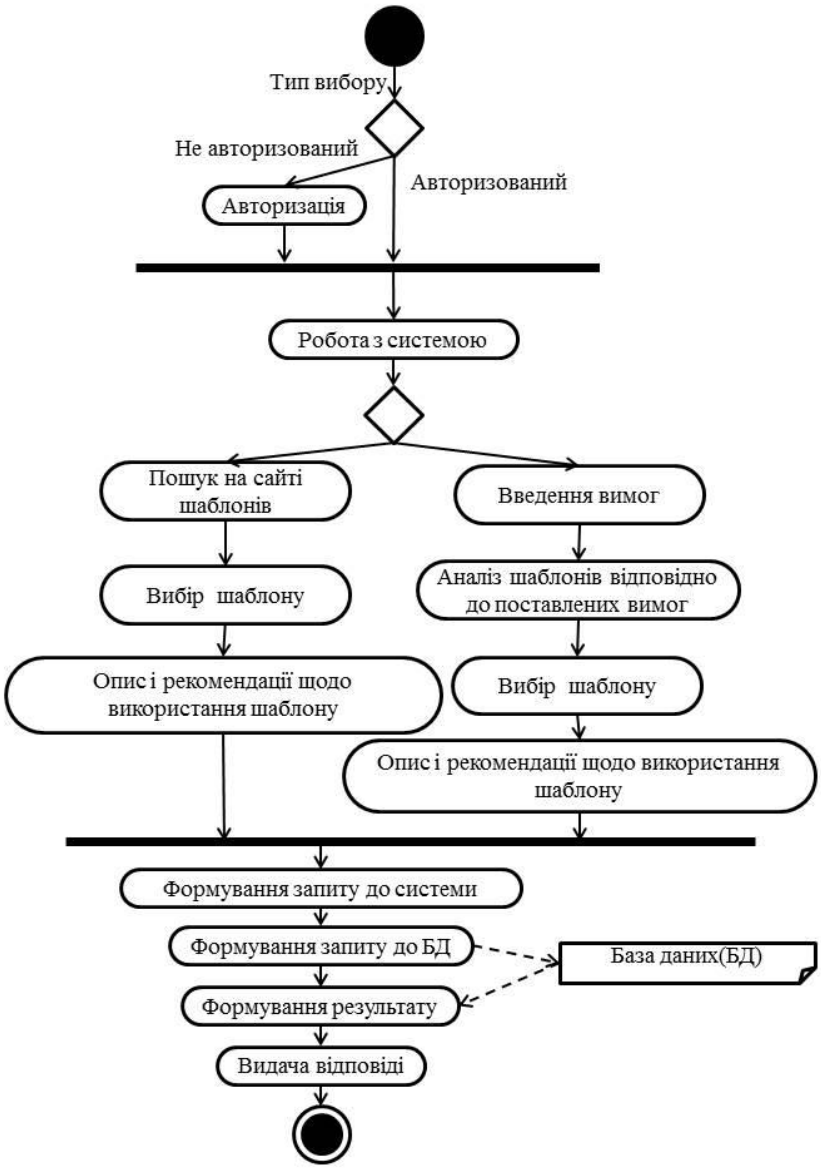

Рис. 2.5 – Діаграма активності

2.3.5 Діаграма розгортання (Deployment Diagram)

Діаграма розгортання — діаграма в UML, на якій відображаються обчислювальні вузли під час роботи програми, компоненти, та об'єкти, що виконуються на цих вузлах. Компоненти відповідають представленню робочих

екземплярів одиниць коду. Компоненти, що не мають представлення під час роботи програми на таких діаграмах не відображаються; натомість, їх можна відобразити на діаграмах компонент. Діаграма розгортання відображає робочі екземпляри компонент, а діаграма компонент, натомість, відображає зв'язки між типами компонент. А тобто, діаграма розгортання служить для моделювання працюючих вузлів і артефактів, розгорнутих на них. Між артефактом і логічним елементом (компонентом), що він реалізує, установлюється залежність маніфестації.

Система позиціонується як веб-сервіс, тому основним апаратним засобом буде Веб-сервер та сервер БД. Користувач має доступ до системи через ПК, під'єднаний до Інтернету. Діаграма активності знаходиться на рисунку 2.6.

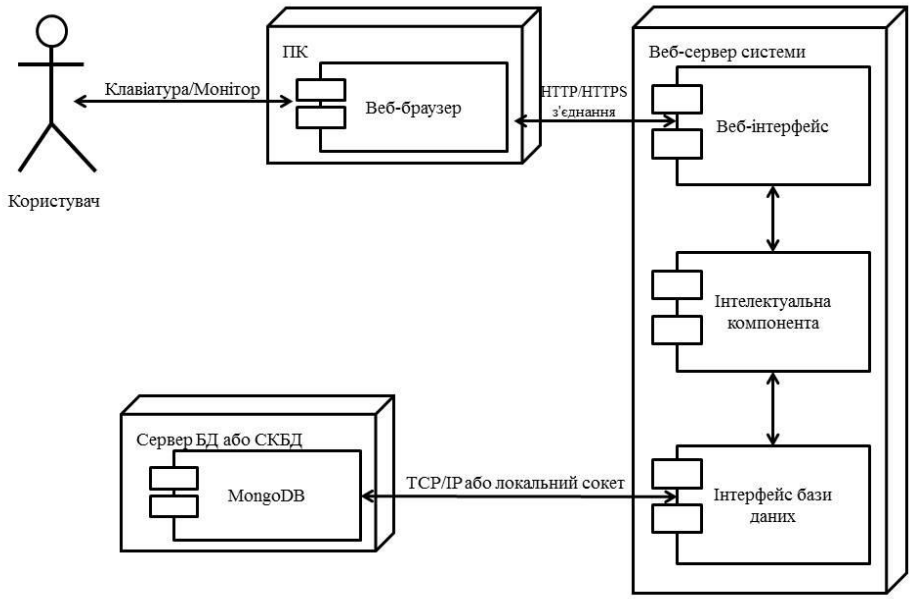

Рис. 2.6 – Діаграма розгортання

Вузли в даній системі: ПК, Веб-сервер, сервер БД або СКБД.

Користувач з ПК який має Веб-браузер через HTTP/HTTPS може з'єднатись з Веб-сервером. Тепер зупинимось детальніше на Веб-сервер та СКБД.

Отже вузол Веб-сервер системи складається з артефактів (відповідно до UML 2.0 нотації) :

- Веб – інтерфейс;

- Інтелектуальна компонента;
- Інтерфейс бази даних.

Веб-інтерфейс – це звичайна Веб-сторінка , що відповідає за діалог системи з користувачем.

Інтелектуальна компонента – відповідальна за правильну інтерпретацію запитів від користувача, видачу даних та обробку даних.

Інтерфейс БД – формує запити до СКБД на основі отриманих від інтелектуальної компоненти критеріїв.

Артефактом СКБД є система управління MongoDB. Дана компонента відповідає за зберігання та доступ до даних.

2.4 Висновок до розділу

Основні результати даного розділу полягають в наступному:

1. Досліджено та обґрунтовано вибір методу формулювання вимог до архітектури ПЗ, який полягає в адаптації методу проектування вимог до архітектури ПЗ, який наведено у [9]. Перевагою даного методу є забезпечення повноти характеристик якості відносно стандарту та задоволеність критеріям якості вимог.
2. Розроблено концептуальну схема процесу детектування шаблонів проектування.
3. Розроблено метод детектування архітектурних патернів програмного забезпечення, який базується на роботі нейронної мережі, що дає змогу забезпечити достовірність результатів детектування і як наслідок дозволяє адекватно встановити відповідність архітектурних шаблонів вимогам доПЗ. Даний формалізований метод є основою інтелектуальної складової програмної системи вибору архітектурного патерну.
4. Побудовані UML діаграми, які зображають модель майбутньої системи і є основою для розробки веб-сайту.

РОЗДІЛ 3
ПОГРАМНИЙ ІНСТРУМЕНТАЛЬНИЙ ЗАСІБ ДЕТЕКТУВАННЯ ПАТЕРНІВ АРХІТЕКТУРИ

3.1 Обґрунтування вибору концепції реалізації програмного інструментального засобу детектування патернів архітектури ПЗ

Штучні нейронні мережі (ШНМ) - математичні моделі , а також їх програмні або апаратні реалізації , побудовані за принципом організації та функціонування біологічних нейронних мереж - мереж нервових клітин живого організму. ШНМ являють собою систему з'єднаних і взаємодіючих між собою простих процесорів (штучних нейронів) . Такі процесори зазвичай досить прості. Кожен процесор подібної мережі має справу тільки з сигналами , які він періодично отримує , і сигналами , які він періодично посилає іншим процесорам . І , тим не менш , будучи з'єднаними в досить велику мережу, такі локально прості процесори разом здатні виконувати досить складні завдання.

З точки зору машинного навчання, нейронна мережа являє собою окремий випадок методів розпізнавання образів, дискримінантного аналізу, методів кластеризації. З математичної точки зору, навчання нейронних мереж - це багатопараметричне завдання нелінійної оптимізації. З точки зору розвитку обчислювальної техніки та програмування , нейронна мережа - спосіб вирішення проблеми ефективного паралелізму. Нейронні мережі не програмуються в звичному сенсі цього слова , вони навчаються . Можливість навчання - одне з головних переваг нейронних мереж перед традиційними алгоритмами . Технічно навчання полягає в знаходженні коефіцієнтів зв'язків між нейронами. У процесі навчання нейронна мережа здатна виявляти складні залежності між вхідними даними і вихідними , а також виконувати узагальнення . Це означає , що в разі успішного навчання мережа зможе повернути вірний результат на підставі даних , які були відсутні в навчальній вибірці , а також неповних і частково перекручених даних.

Обчислювальні системи, засновані на штучних нейронних мережах, мають ряд якостей, які відсутні в машинах з архітектурою фон Неймана:

- Масовий паралелізм;
- Розподілене представлення інформації і обчислення;
- Здатність до навчання та узагальненню;
- Адаптивність;
- Властивість контекстуальної обробки інформації;
- Толерантність до помилок;
- Низьке енергоспоживання.

ШНМ мають велику область застосувнь, наприклад прогнозуванням результату чи розпізнаванням образів, розглянемо деякі з них.

Розпізнавання образів і класифікація, в якості образів можуть виступати різні за своєю природою об'єкти: символи тексту, зображення , зразки звуків і т. д. При навчанні мережі пропонуються різні зразки образів із зазначенням того, до якого класу вони відносяться. Як правило представляється як вектор значень ознак. При цьому сукупність всіх ознак повинна однозначно визначати клас, до якого належить зразок. У разі, якщо ознак недостатньо, мережа може співвіднести один і той же зразок з кількома класами, що невірно. Після закінченні навчання мережі їй можна пред'являти невідомі раніше образи і отримувати відповідь про приналежність до певного класу.

Топологія такої мережі характеризується тим, що кількість нейронів у вихідному шарі, як правило, дорівнює кількості визначених класів. При цьому встановлюється відповідність між виходом нейронної мережі і класом, який він представляє. Коли мережі пред'являється якийсь образ, на одному з її виходів повинен з'явитися ознака того, що образ належить цьому класу . У той же час на інших виходах повинен бути ознака того , що образ даному класу не належить. Якщо на двох або більше виходах є ознака приналежності до класу, вважається, що мережа «не впевнена» у своїй відповіді .

Прийняття рішень і керування – це завдання близька до задачі класифікації. Класифікації підлягають ситуації , характеристики яких надходять на вхід нейронної мережі. На виході мережі при цьому повинен з'явитися ознака рішення, яке вона прийняла. При цьому в якості вхідних сигналів використовуються різні критерії опису стану керованої системи[22].

Кластеризація, під якою розуміють розбиття безлічі вхідних сигналів на класи, при тому, що ні кількість, ні ознаки класів заздалегідь не відомі. Після навчання така мережа здатна визначати, до якого класу належить вхідний сигнал. Мережа також може сигналізувати про те, що вхідний сигнал не відноситься ні до одного з виділених класів - це є ознакою нових, відсутніх у навчальній вибірці, даних. Таким чином, подібна мережа може виявляти нові, невідомі раніше класи сигналів. Відповідність між класами, виділеними мережею, і класами, існуючими в предметній області, встановлюється людиною. Представником кластеризацію в ШНМ є нейронні мережі Кохонена.

Нейронні мережі в простому варіанті Кохонена не можуть бути величезними, тому їх ділять на гіпершари (гіперколонки) і ядра (мікроколонки). Ці шари в свою чергу складають гіпершари (гіперколонки), в яких від 500 до 2000 мікроколонок (ядер). При цьому кожен шар ділиться на безліч гіперколонок пронизуючих наскрізь ці шари. Якщо потрібно, то зайві шари і нейрони видаляються або додаються. Така система дозволяє нейронним мережам бути пластичними.

Прогнозування, здібності нейронної мережі до прогнозування безпосередньо випливають з її здатності до узагальнення і виділенню прихованих залежностей між вхідними та вихідними даними. Після навчання мережа здатна передбачити майбутнє значення якоїсь послідовності на основі декількох попередніх значень і якихось існуючих в даний момент факторів. Слід зазначити, що прогнозування можливе тільки тоді, коли попередні зміни дійсно в якійсь мірі зумовлюють майбутні. Наприклад, прогнозування цін акцій на основі цін за минулий тиждень може виявитися успішним, тоді як прогнозування результатів завтрашньої лотереї на основі даних за останні 50 років майже напевно не дасть жодних результатів .

Апроксимація, нейронні мережі можуть апроксимувати неперервні функції. Доведена узагальнена апроксимаційна теорем[23]: за допомогою лінійних операцій і каскадного з'єднання можна з довільного нелінійного елемента отримати пристрій, котри зможе вичислити будь-яку неперервну функцію з деякою наперед заданою точністю. Це означає, що нелінійна характеристика нейрона може бути довільною: від сигмоїдальної до довільного хвильового пакета чи вейвлета, синуса або многочлена. Від вибору нелінійної функції може залежати складність конкретної мережі, але з будь-якою нелінійністю мережа залишається універсальним аппроксиматором і при правильному виборі структури може досить точно апроксимувати функціонування будь-якого безперервного автомата.

Стиснення даних і асоціативна пам'ять – здатність нейромереж до виявлення взаємозв'язків між різними параметрами дає можливість висловити дані великої розмірності більш компактно, якщо дані тісно взаємозалежні один з одним. Зворотний процес - відновлення вихідного набору даних з частини інформації – називається асоціативною пам'яттю. Асоціативна пам'ять дозволяє також відновлювати вихідний сигнал, образи з зашумленням, пошкоджені вхідні дані. Рішення завдання гетероасоціативної пам'яті дозволяє реалізувати пам'ять, адресовану по вмісту.

3.2 Обґрунтування технології реалізації програмного засобу

Повнота програмних та користувацьких інтерфейсів значною мірою залежить від технології, що використовується для їх розробки. Сучасні технології розробки досягли значного рівня розвитку та різноманіття підходів до проектування, реалізації та інтеграції компонентів систем. Побудова настільних клієнтських додатків на сьогодні досягла достатньо високого рівня розвитку із наданні інтерактивних, зручних та ефективних засобів як для розробки програмних систем так і для використання результатів їх роботи.

Для вибору найбільш оптимальної технології реалізації необхідно виконати огляд та порівняння характеристик для поширених та актуальних на сьогодні платформ і технологій в рамках підтримки мережно-орієнтованої парадигми. Найпоширенішими на сьогодні технологічними платформами розробки настільних додатків є технології Java, MFC, Qt та .NET.

Насамперед необхідно визначити вимоги, щодо технології реалізації, які необхідно задовольнити в процесі розробки програмного засобу для забезпечення рівня функціональної повноти, надійності, безпеки та інших показників:

- Підтримка мережних протоколів та сервісів для доступу до сховища даних
- Хороша інтеграція із СКБД для підвищення ефективності роботи з даними
- Наявність достатньої кількості візуальних компонентів, особливо для роботи по редагуванню та відображенню даних
- Інтеграція із клієнтською операційною системою (Windows) з метою підвищення швидкодії виконання коду
- Надійність середовища виконання коду, механізмів контролю помилок та перехоплення виняткових ситуацій
- Можливість автоматизації процесу розробки з метою зниження часових витрат
- Наявність та актуальність супровідної документації

Виконаємо огляд характеристик сучасних технологій розробки настільних додатків та їх порівняння у відповідності рівня задоволення вимог що висуваються до мережно-орієнтованої системи.

Java — платформа виконання для однойменної об'єктно-орієнтованої мови програмування, що є її основним компонентом. Випущена компанією Sun Microsystems у 1995 році. Синтаксис мови Java багато в чому походить від C та C++. У офіційній реалізації, Java програми компілюються у байткод, який при виконанні інтерпретується віртуальною машиною для конкретної платформи. Основною

відміною рисою платформи Java є її орієнтація на різноманітні апаратні та програмні платформи із збереженням усіх концепцій розробки. Це значною мірою ускладнює глибоку інтеграцію із цільовою системою та дозволяє використовувати лише неспецифічні компоненти, що знижує швидкодію та надійність системи.

Мова значно запозичила синтаксис із C і C++. Зокрема, взято за основу об'єктну модель C++, проте її модифіковано. Усунуто можливість появи деяких конфліктних ситуацій, що могли виникнути через помилки програміста та полегшено сам процес розробки об'єктно-орієнтованих програм. Ряд дій, які в C/C++ повинні здійснювати програмісти, доручено віртуальній машині. Передусім, Java розроблялась як платформо-незалежна мова, тому вона має менше низькорівневих можливостей для роботи з апаратним забезпеченням. За необхідності таких дій java дозволяє викликати підпрограми, написані іншими мовами програмування.

Основна перевага використання байт-коду — це портативність. Тим не менш, додаткові витрати на інтерпретацію означають, що інтерпретовані програми будуть майже завжди працювати повільніше, ніж скомпільовані у машинний код, і саме тому Java одержала репутацію «повільної» мови. Проте, цей розрив суттєво скоротився після введення декількох методів оптимізації у сучасних реалізаціях JVM.

Qt - кросплатформний інструментарій розробки ПЗ мовою програмування C++, що базується на певному наборі бібліотек, які забезпечують програмну обгортку навколо стандартного API системи. Використання Qt покликане зменшити складність розробки клієнтських інтерфейсів за рахунок зниження платформної залежності C++ та розширення функціональності стандартного набору компонентів API операційних систем.

Дозволяє запускати написане за його допомогою ПЗ на більшості сучасних операційних систем шляхом простої компіляції програми для кожної ОС без зміни початкового коду. Включає в себе всі основні класи, які можуть бути потрібні при розробці прикладного програмного забезпечення, починаючи з елементів графічного інтерфейсу і закінчуючи класами для роботи з мережею, базами даних і XML. Qt є

повністю об'єктно-орієнтованим, легко розширюваною і які підтримують техніку компонентного програмування.

До версії 4.0.0 під вільною ліцензією розповсюджувалися лише Qt / Mac, Qt/X11, Qt / Embedded, але, починаючи з 4.0.0 (випущеної в кінці 2005), Qt Software «звільнили» і Qt / Windows. Слід зазначити, що існували сторонні вільні версії Qt / Windows і 4.0.0, зроблені на основі Qt/X11. З часу своєї появи в 1996 році бібліотека Qt лягла в основу тисяч успішних проектів у всьому світі. Крім того, Qt є фундаментом популярного робочого середовища KDE, що входить до складу багатьох дистрибутивів Linux

Відмінна особливість Qt від інших бібліотек - використання Meta Object Compiler (MOC) - попередньої системи обробки початкового коду (загалом, Qt - це бібліотека не для чистого C + +, а для його особливого діалекту, з якого і «перекладає» MOC для подальшої компіляції будь-яким стандартним C + + компілятором). MOC дозволяє у багато разів збільшити потужність бібліотек, вводячи такі поняття, як слоти і сигнали. Крім того, це дозволяє зробити код більш лаконічним. Утиліта MOC шукає в заголовних файлах на C + + опису класів, що містять макрос Q_OBJECT, і створює додатковий вихідний файл на C + +, що містить мета-об'єктний код.

Microsoft Foundation Classes – набір бібліотек для розробки графічного інтерфейсу користувача для операційної системи Windows із використанням Visual C++. MFC являє собою оболонку навколо низькорівневого API операційної системи Windows, додаючи рівень абстракції для спрощення розробки програмних додатків.

Перша версія MFC була випущена разом з сьомою версією 16-розрядного компілятора мови C/C++ компанії Microsoft в 1992 році. Для тих, хто займався розробкою програм з використанням API функцій, пакет MFC обіцяв вельми значне підвищення продуктивності процесу програмування.

MFC не забезпечує компонування елементів управління всередині вікна і це створює проблеми при бажанні зробити вікно змінюваного розміру. Потрібно вручну переміщати елементи керування при зміні розмірів вікна (це пояснює, чому

більшість діалогових вікон Windows незмінного розміру). Ця проблема набуває більшого значення для програм, інтерфейс яких повинен бути переведений на мову з більш довгими словами або реченнями. Потрібно перекомпонувати вікна для кожної такої мови.

Підтримка платформи MFC проводиться і досі, вона входить в пакет поставки середовища Visual Studio 2010, але її популярність та вдосконалення значно знизились із виходом платформи .NET, що володіє суттєвими перевагами.

Платформа .NET являє собою технологію та середовище виконання, що розроблені компанією Microsoft задоволення зростаючих потреб в сфері розробки прикладних додатків. Порівняно із іншими існуючими на даний момент технологіями .NET володіє певними особливостями, що дозволяють посідати їй одне з перших місць по популярності в усіх сферах розробки. Розглянемо ці особливості:

- Об’єктно – орієнтована модель платформи. Середовище .NET Framework та мова програмування C# з моменту заснування базуються на повністю об’єктно – орієнтованій моделі. Всі сутності представлені в вигляді класів, інтерфейсів та зв’язків між ними. Це забезпечує хорошу абстракцію та легку подальшу модернізацію коду.
- Хороший дизайн класів. Бібліотека базових класів .NET містить потужний набір відлагоджених та оптимізованих класів для пришвидшення розробки програмних рішень
- Незалежність від мови програмування. Гарантує можливість модернізації за рахунок включення бібліотек інших розробників, можливість паралельної розробки модулів на різних мовах програмування.
- Ефективність доступу до даних. Середовище .NET містить в своєму складі технологію ADO.NET, яка створена для спрощення процесу доступу до даних в різноманітних вмістилищах, встановлення надійних, швидкодіючих з’єднань із базами даних. Також класи технології ADO.NET спрощують маніпуляції над даними та їх обробку і збереження.

- Розділення коду. Платформа .NET володіє покращеним механізмом збереження коду різного призначення. Так, наприклад, код, що відповідає за інтерфейс форми зберігається в файлах-дизайнерах, а програмний код, що гарантує роботу форми – в програмних файлах. Це спрощує як написання коду, так і його подальшу модернізацію.
- Підвищення безпеки. Всі збірки програм, що виконані на основі засобів платформи .NET містять внутрішню інформацію про режими роботи програми, її процеси, користувачі та їх привілегії. Ці дані дозволяють контролювати процес роботи програми та ефективно його змінювати в разі потреби.
- Visual Studio 2010. .NET інтегрується із середовищем розробки Visual Studio 2010. Дане середовище містить засоби автоматизації розробки програм, об’єднує в собі більшість сучасних технологічних розробок компанії Microsoft, що дозволяє розробляти дійсно якісний програмний код.
- C# є мовою програмування, що спочатку була створена для використання, як основна мова програмування платформи .NET. Дана мова програмування володіє прозорим, простим та гнучким синтаксисом, підтримкою усіх аспектів платформи .NET, покращеною підтримкою в середовищі розробки Visual Studio 2010, потужною реалізацією об’єктно – орієнтованої парадигми.

Можливості платформи .NET по створенню графічного інтерфейсу користувача та засоби доступу до даних є надзвичайно розвинутими. Провідною технологією для розробки користувацьких інтерфейсів в середовищі операційної системи Windows є Windows Forms і в вебі ASP. Microsoft повністю перебудувала ASP.NET, ґрунтуючись на CLR, який є основою всіх застосунків Microsoft .NET. Розробники можуть писати код для ASP.NET, використовуючи практично будь-які мови програмування, що входять у комплект .NET Framework. ASP.NET має перевагу у швидкості в порівнянні зі скриптовими технологіями, тому що при першому зверненні код компілюється і поміщається в спеціальний кеш, і згодом тільки виконується, не вимагаючи витрат часу на парсинг, оптимізацію, і т. д.

На основі наведених характеристик технологій проведемо аналіз задоволення рівня вимог, що висуваються до реалізації мережно-орієнтованої системи.

Таблиця 3.1

Порівняльний аналіз технологій реалізації прикладних додатків

	Java	Qt	MFC	.NET
Підтримка мережних протоколів	Висока	Середня	Середня	Висока
Інтеграція із СКБД	Висока, концепція драйверів JDBC	Висока	Середня	Висока, концепція провайдерів, повна – для SQL Server
Візуальні компоненти	Стандартні загальні набори	Узагальнені набори із хорошою функціональністю	Великий набір компонент	Великий набір стандартних компонент, розширюваний каркас користувацьких компонентів
Інтеграція із ОС Windows	Слабка	Середня	Висока	Висока
Надійність	Середня (можливі витоки пам’яті)	Середня	Середня	Висока (автоматичний збір «сміття», захищений режим виконання)

Продовження табл. 3.1

	Java	Qt	MFC	.NET
Рівень автоматизації процесу розробки	Розвинуті середовища розробки	Розвинуті середовища розробки	Використання діалог-майстер, але велика кількість ручного кодування	Повнофункціональне інтегроване середовище розробки
Супровідна документація	Web-формат, більшість англійською мовою	Повна та доступна документація	Повна та доступна документація	Повна та доступна документація

На основі проведеного аналізу у зв’язку із потребою задоволення усіх вимог до реалізації системи контролю товарообігу було обрано технологію .NET для реалізації інтерфейсу користувача та доступу до даних у репозиторіях.

3.3 Обґрунтування вибору СКБД

У сфері високих технологій серед СКБД існує величезна різноманітність варіацій концепцій, архітектури та реалізацій ПЗ для обслуговування процесів управління даними. На основі критеріїв розповсюдженості, технічної завершеності та документованості було відібрано 3 засоби-кандидати: MySQL 5.1, MongoDB та MS SQL Server 2008.

Microsoft SQL Server - це система управління базами даних, що забезпечує мережевий багатокористувацький доступ та використовує розширену мова запитів T-SQL. Веде свою історію з 1989 року, перша версія була створена на основі Sybase.

У попередній версії 2005 була введена підтримка CLR, яка дозволяла писати процедури з використанням мов, що працюють на платформі. Net. Останньою версією даного СУБД є SQL Server 2008. SQL Server 2008 спрямований на те, щоб зробити керування даними самоналагоджувальний, самообслуговуваним і самопідтримуваним механізмом - для реалізації цих можливостей були створені технології SQL Server Always On. Це дозволить зменшити до нуля час знаходження сервера в неробочому стані.

Microsoft SQL Server Express є безкоштовною версією SQL Server, розвитком системи MSDE. Дана версія має деякі технічні обмеження, але вона цілком годиться для ведення програмних комплексів у масштабах невеликої компанії. Містить повноцінну підтримку нових типів даних, у тому числі XML-специфікації. Фактично, це повноцінний MS SQL Server, включаючи всі його компоненти програмування, підтримку національних алфавітів і Unicode.

Немає жодних перешкод для подальшого розгортання накопиченої бази даних на MS SQL Server версії відмінною від Express. Управління сервером СУБД MS SQL Server Express здійснюється через утиліту Management Studio, так само як і комерційними версіями даної СУБД.

MySQL — вільна система керування реляційними базами даних.

Ця система керування базами даних (СКБД) з відкритим кодом була створена як альтернатива комерційним системам. MySQL з самого початку була дуже схожою на mSQL, проте з часом вона все розширювалася і зараз MySQL — одна з найпоширеніших систем керування базами даних. Вона використовується, в першу чергу, для створення динамічних веб-сторінок, оскільки має чудову підтримку з боку різноманітних мов програмування.

MySQL — компактний багатопоточний сервер баз даних. Характеризується великою швидкістю, стійкістю і простотою використання. MySQL був розроблений компанією «ТсХ» для підвищення швидкодії обробки великих баз даних.

MySQL вважається гарним рішенням для малих і середніх застосувань. Вихідні коди сервера компілюються на багатьох платформах. Найповніше

можливості сервера виявляються в UNIX-системах, де є підтримка багатопоточності, що підвищує продуктивність системи в цілому.

Для некомерційного використання MySQL є безкоштовним. Можливості сервера MySQL:

- простота у встановленні та використанні;
- підтримується необмежена кількість користувачів, що одночасно працюють із БД;
- кількість рядків у таблицях може досягати 50 млн.;
- висока швидкість виконання команд;
- наявність простої і ефективної системи безпеки.

Водночас головними недоліками MySQL є:

- Платність при комерційному використанні
- Неповна функціональність відносно сучасних вимог реляційної алгебри (наприклад, зовнішні ключі потребують встановлення додаткових компонентів, запити між базами даних неможливі або надзвичайно утруднені)
- Менша надійність роботи при значному навантаженні
- Слабка інтеграція із сучасними платформами розробки

MongoDB — документо-орієнтована система керування базами даних (СКБД) з відкритим вихідним кодом, яка не потребує опису схеми таблиць. MongoDB займає нішу між швидкими і масштабованими системами, що оперують даними у форматі ключ/значення, і реляційними СКБД, функціональними і зручними у формуванні запитів.

MongoDB підтримує зберігання документів в JSON-подібному форматі, має досить гнучку мову для формування запитів, може створювати індекси для різних збережених атрибутів, ефективно забезпечує зберігання великих бінарних об'єктів, підтримує журналювання операцій зі зміни і додавання даних в БД, може працювати відповідно до парадигми Map/Reduce, підтримує реплікацію і побудову відмовостійких конфігурацій. У MongoDB є вбудовані засоби із забезпечення

шардінга (розподіл набору даних по серверах на основі певного ключа), комбінуючи який реплікацією даних можна побудувати горизонтально масштабований кластер зберігання, в якому відсутня єдина точка відмови (збій будь-якого вузла не позначається на роботі БД), підтримується автоматичне відновлення після збою і перенесення навантаження з вузла, який вийшов з ладу. Розширення кластера або перетворення одного сервера в кластер проводиться без зупинки роботи БД простим додаванням нових машин.

Основні можливості MongoDB:

- Документо-орієнтоване сховище (проста та потужна JSON - подібна схема даних)
- Досить гнучка мова для формування запитів
- Динамічні запити
- Повна підтримка індексів
- Профілювання запитів
- Швидкі оновлення «на місці»
- Ефективне зберігання двійкових даних великих обсягів, наприклад, фото та відео
- Журналювання операцій, що модифікують дані в БД
- Підтримка відмовостійкості і масштабованості: асинхронна реплікація, набір реплік і шардінг
- Може працювати відповідно до парадигми MapReduce

Важливу роль у підвищенні продуктивності системи грає інтеграція між середовищем розробки, середовищем виконання і сховищами даних. Повільне постачання даних у швидкий програмний код буде гальмувати загальний хід виконання програми, зменшувати реакцію системи, підвищувати накладні витрати. У той же час слабкий механізм відображення даних на логічні конструкції коду може спричинити ускладнення їх обробки і подальшого збереження.

В результаті аналізу функціональних вимог до програмної системи та сховища даних, предметної області, потреб користувачів було обрано СКБД MongoDB як одночасно економічно вигідний та мультифункціональний продукт.

3.4 Розробка моделі бази даних

Для опису ПО використовується концептуальна модель ПО. Вона є формальним зображення сукупності думок, які характеризують можливі стани ПО, а також переходи з одного стану в інший. Наступним кроком є побудова концептуальної схеми. Концептуальна схема – фіксація концептуальної моделі ПО засобами конкретних мов моделей даних. Існує багато мов, які претендують на роль мов концептуального моделювання ПО.

Оскільки мето метою дипломної роботи є створення програмної системи для вибору архітектурних шаблонів потрібно врахувати всі аспекти його роботи. Проаналізувавши дану предметну область виявлено основні сутності та атрибути.

Кожна сутність відображається на окрему таблицю, сутності супертипів та підтипів можуть розділятися на кілька таблиць, атрибут перетворюється на окремий стовбець із обранням певного типу. Усі складові певного унікального атрибута сутності стають складовими первісного ключа таблиці, якщо ж у сутності неможливо виділити однозначний унікальний ідентифікатор то вводиться синтетичний первісний ключ для таблиці. Зв’язки між сутностями породжують зовнішні ключі таблиць. Факультативним зв’язкам відповідають NULL- допустимі значення, а обов’язковим – NOT NULL стовпці.

Фізичну реалізацію ER- моделі зручно представити у табличному вигляді із наведенням імені таблиці та відповідної сутності, назвами та типами стовпців.

Таблиця 3.2

Фізична реалізація сутності «Проекту»

	Назва сутності: «Проект» Назва таблиці: Project		
Назва стовпця	Тип значення атрибуту	Тип зв’язку	Примітка
ID	int	NOT NULL	PK
Name	nvarchar	NOT NULL	
CreateDataTime	datetime	NULL	
UserId	int	NOT NULL	FK

Таблиця 3.3

Фізична реалізація сутності «Модель вимоги»

	Назва сутності: «Модель вимоги» Назва таблиці: RequirementModel		
Назва стовпця	Тип значення атрибуту	Тип зв’язку	Примітка
ID	int	NOT NULL	PK
Name	nvarchar	NOT NULL	
CreateDataTime	datetime	NULL	
UserId	int	NOT NULL	FK
Comment	nvarchar	NULL	
Project_ID	Int	NOT NULL	FK

Таблиця 3.4

Фізична реалізація сутності «Вимоги»

	Назва сутності: «Вимоги» Назва таблиці: Requirement		
Назва стовпця	Тип значення атрибуту	Тип зв’язку	Примітка
ID	int	NOT NULL	PK
Name	nvarchar	NOT NULL	
Description	nvarchar	NULL	
Target	nvarchar	NULL	
AcceptProcedure	nvarchar	NULL	
MarkInterpretation	nvarchar	NULL	
ScaleType	int	NULL	
ControlFunction	nvarchar	NOT NULL	
Weight	float	NULL	
RequirementModel_ID	Int	NOT NULL	FK

Таблиця 3.5

Фізична реалізація сутності «Характеристика»

	Назва сутності: «Характеристика» Назва таблиці: Characteristics		
Назва стовпця	Тип значення атрибуту	Тип зв’язку	Примітка
ID	int	NOT NULL	PK
Name	nvarchar	NOT NULL	

Таблиця 3.6

Фізична реалізація сутності «Під характеристика»

	Назва сутності: «Під характеристика» Назва таблиці: SubCharacteristics		
Назва стовпця	Тип значення атрибуту	Тип зв’язку	Примітка
ID	int	NOT NULL	PK
Name	nvarchar	NOT NULL	
Characteristic_ID	int	NOT NULL	FK

Таблиця 3.7

Фізична реалізація сутності «Значення шкали»

	Назва сутності: «Значення шкали» Назва таблиці: ScaleValue		
Назва стовпця	Тип значення атрибуту	Тип зв’язку	Примітка
ID	int	NOT NULL	PK
Value	nvarchar	NOT NULL	
Requirement_ID	int	NOT NULL	FK

Таблиця 3.8

Фізична реалізація сутності «Тип збору оцінки»

	Назва сутності: «Тип збору оцінки» Назва таблиці: GatheringType		
Назва стовпця	Тип значення атрибуту	Тип зв’язку	Примітка
ID	int	NOT NULL	PK
Value	nvarchar	NOT NULL	
Requirement_ID	int	NOT NULL	FK

Таблиця 3.9

Фізична реалізація сутності «Атрибути вимог»

	Назва сутності: «Атрибути вимог» Назва таблиці: AttributeReq		
Назва стовпця	Тип значення атрибуту	Тип зв’язку	Примітка
ID	int	NOT NULL	PK
Value	nvarchar	NOT NULL	
Requirement_ID	int	NOT NULL	FK
AttributeTypeReq_ID	int	NULL	FK

Таблиця 3.10

Фізична реалізація сутності «Типи атрибутів вимог»

	Назва сутності: «Типи атрибутів вимог» Назва таблиці: AttributeTypeReq		
Назва стовпця	Тип значення атрибуту	Тип зв’язку	Примітка
ID	int	NOT NULL	PK
Name	nvarchar	NOT NULL	

Таблиця 3.11

Фізична реалізація сутності «Тип шаблону»

	Назва сутності: «Тип шаблону» Назва таблиці: PatternType		
Назва стовпця	Тип значення атрибуту	Тип зв’язку	Примітка
ID	int	NOT NULL	PK
Name	nvarchar	NOT NULL	

Таблиця 3.12

Фізична реалізація сутності «Модель шаблону»

	Назва сутності: «Модель шаблону» Назва таблиці: PatternModel		
Назва стовпця	Тип значення атрибуту	Тип зв’язку	Примітка
ID	int	NOT NULL	PK
Name	nvarchar	NOT NULL	
DateTime	datetime	NULL	

Таблиця 3.13

Фізична реалізація сутності «Шаблон»

	Назва сутності: «Шаблон» Назва таблиці: Pattern		
Назва стовпця	Тип значення атрибуту	Тип зв’язку	Примітка
ID	int	NOT NULL	PK
Name	nvarchar	NOT NULL	
Description	nvarchar	NULL	
Sample	nvarchar	NULL	
PatternModel_ID	int	NOT NULL	FK
PatternType_ID	int	NOT NULL	FK

Таблиця 3.14

Фізична реалізація сутності «Атрибути шаблонів»

	Назва сутності: «Атрибути шаблонів» Назва таблиці: AttributePat		
Назва стовпця	Тип значення атрибуту	Тип зв’язку	Примітка
ID	int	NOT NULL	PK
Value	nvarchar	NOT NULL	
Pattern_ID	int	NOT NULL	FK
AttributeType_ID	int	NULL	FK

Для відображення структури бази даних наведемо рисунок 3.1:

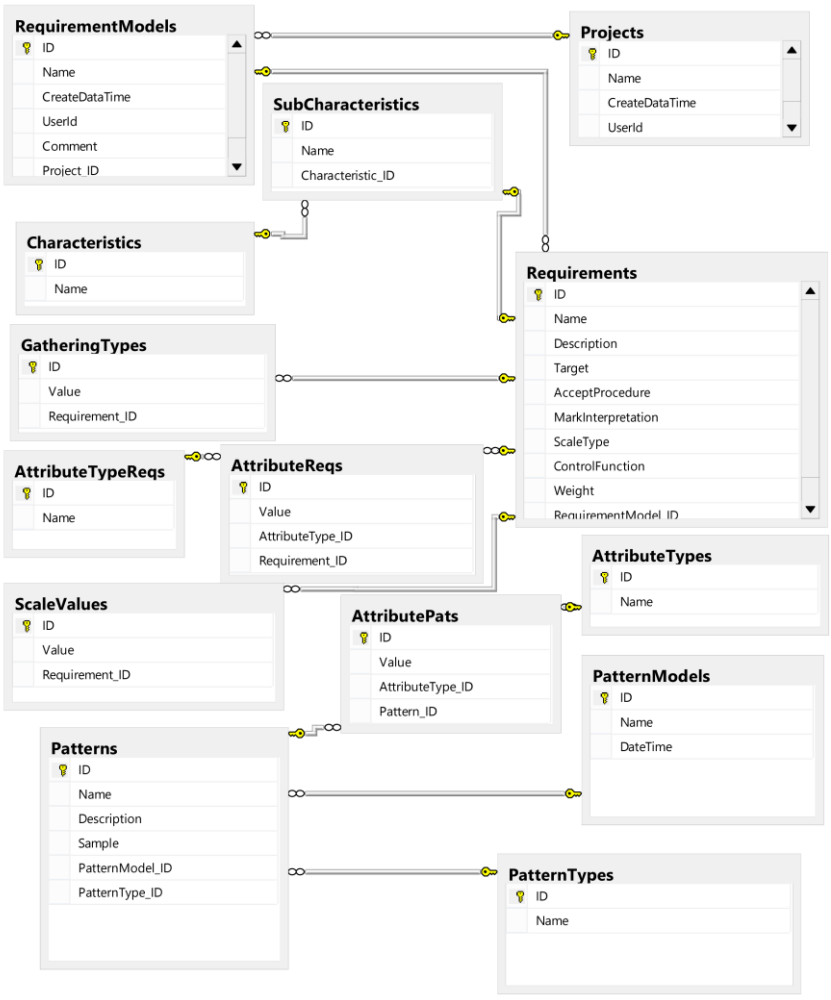

Рис 3.1 – Структурна схема бази даних

3.5 Реалізація модуля детектування шаблонів проектування

Для програмної реалізації нейромережі використано бібліотеку NeuronDotNet, яка дозволяє гнучко конструювати різноманітні нейромережі із використанням мови C#.

Для формування та навчання нейромережі розроблено модуль NeuralNetworkBuilder, що містить статичний метод BuildAndTrainNetwork. В процесі формування нейромережі конструюються вхідний та вихідний рівень нейронів, нейрони прихованих рівнів та встановлюються зв’язки між ними (лістинг 4.1).

Лістинг 3.1 – Конструювання нейромережі

```
        var inputCount = trainingModel.TrainingModels.Max(x =>
x.Key.InternalRequirementProperties.Count);
                    var outputCount = trainingModel.TrainingModels.Max(x =>
x.Value.PatternModels.Sum(pm=>pm.PropertiesLevels.Count));
                   LinearLayer inputLayer = new LinearLayer(inputCount);
                    SigmoidLayer outputLayer = new SigmoidLayer(outputCount);
                    var hiddenLayers = new List<SigmoidLayer>();
                    for (int i = 0; i < outputCount; i++)
                    {
                        var hiddenLayer = new SigmoidLayer(outputCount);
                        hiddenLayers.Add(hiddenLayer);
                        if (i == 0)
                        {
                            new BackpropagationConnector(inputLayer, hiddenLayer);
                            continue;
                        }
                        var prevLevel = hiddenLayers[i - 1];
                        new BackpropagationConnector(prevLevel, hiddenLayer);
                    }
                    var hl = hiddenLayers.Last();
                    new BackpropagationConnector(hl, outputLayer);

                    BackpropagationNetwork network = new
BackpropagationNetwork(inputLayer,outputLayer);
```

Для навчання нейромережі використовується попередньо сформована множина вимог та детектованих шаблонів проектування, навчання проводиться без вчителя (лістинг 3.2).

Лістинг 3.2 – Навчання нейромережі

```
        TrainingSet ts = new TrainingSet(inputCount, outputCount);
                    foreach (var tm in trainingModel.TrainingModels)
                    {
                        var inputVector =
                            tm.Key.InternalRequirementProperties.OrderBy(x =>
x.InternalID).Select(x => x.Value).ToArray();
                        var outputVector =
                            tm.Value.PatternModels.OrderBy(x =>
x.PatternModel.InternalID)
                                .SelectMany(x => x.PropertiesLevels.OrderBy(kvp =>
kvp.Key).ToList()).Select(x => x.Value)
                                .ToArray();
                        var sample = new TrainingSample(inputVector, outputVector);
                        ts.Add(sample);
                    }
                    network.Learn(ts,trainingModel.TrainingCyclesCount);
```

Після навчання нейромережа може використовуватись для пошуку шаблонів із допомогою методу Run (лістинг 3.3). На вхід методу подається множина значень атрибутів моделі вимог.

Лістинг 3.3 – Пошук шаблонів з допомогою нейромережі

```
        public double[] Run(InternalRequirement inputRequirement)
                {
                    var input =
inputRequirement.InternalRequirementProperties.OrderBy(x => x.InternalID).Select(x =>
x.Value).ToArray();
                    var output = _network.Run(input);
                    return output;
                }
```

Результуючий вектор значень аналізується із використанням методу простого ковзного середнього для пошуку імовірного шаблону проектування (лістинг 4.4).

Лістинг 3.4 – Аналіз результуючого вектора та пошук шаблону

```
        public PatternModel ResolvePattern(double[] valuesVector)
                {
                    var deltas = new double[_patternsList.Count];

                    var startIndex = 0;
                    for (int i = 0; i < _patternsList.Count; i++)
                    {
                        var pattern = _patternsList[i];
```

```
                        var patternVector =
pattern.PatternModelProperties.OrderBy(x=>x.InternalID).Select(x =>
x.Value).ToArray();
                        var targetVector =
valuesVector.Skip(startIndex).Take(patternVector.Length).ToArray();
                        startIndex += targetVector.Length;
                        for (int j = 0; j < patternVector.Length; j++)
                        {
                            patternVector[j] = Math.Abs(patternVector[j] -
targetVector[j]);
                        }
                    deltas[i] = patternVector.Sum()/(double)patternVector.Length;
                }

                var minDelta = deltas.Min();
                return _patternsList[Array.IndexOf(deltas, minDelta)];
            }
```

Запропонована концептуальна модель та програмна реалізація для детектування шаблонів проектування на основі моделей вимог дозволяє гнучко змінювати підходи до формування моделей вимог та шаблонів, налаштовувати параметри аналізу вимог за допомогою зміни конфігурації нейромережі чи умов її навчання. Крім того, використання нейромережі як основного компонента аналізатора дозволяє забезпечити прогресивність параметрів пошуку, підвищити точність детектування, автоматизувати процес пошуку шаблонів проектування.

3.6 Представлення користувацького інтерфейсу програмної системи

Користувацький інтерфейс програмної системи для вибору архітектурного шаблону повинна забезпечувати зручне введення, редагування та перегляд інформації, бути інтуїтивно зрозумілим, забезпечувати перевірку коректності вводу даних.

Для реалізації алгоритму роботи програми запропоновано використати мультивіконний інтерфейс із головною та дочірніми формами. Інтерфейс також обладнано стрічкою меню, що полегшує навігацію між вікнами програми (рис. 4.4).

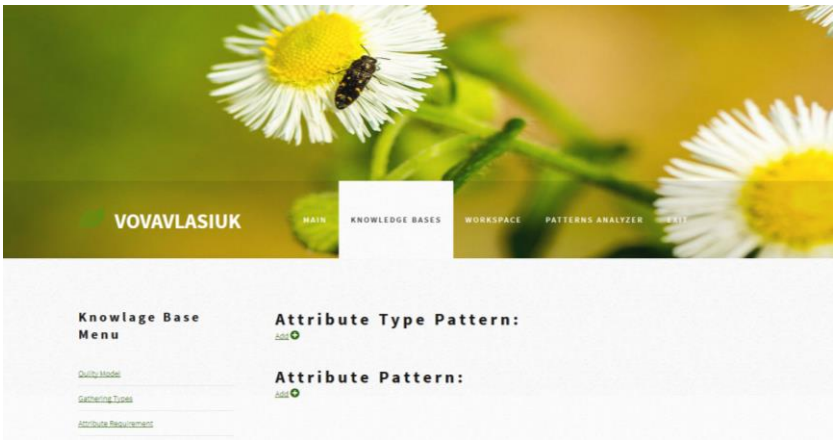

Рис. 4.4 – Основне вікно та вигляд меню

Але для користування системою користувач спочатку повинен зареєструватися, вдівши своє ім., електрону пошту, пароль і павтор паролю, мобільний телефон(рис. 4.4). Після введення всіх параметрів користувачу потрібно ввести капчу і підтвердити згідність з правилами системи. При невірному введені даних користувачу відобразиться попередження(рис. 4.5). По завершені реєстрації користувачу на електрону пошту прийде лист з іменем і паролем до веб-сайту.

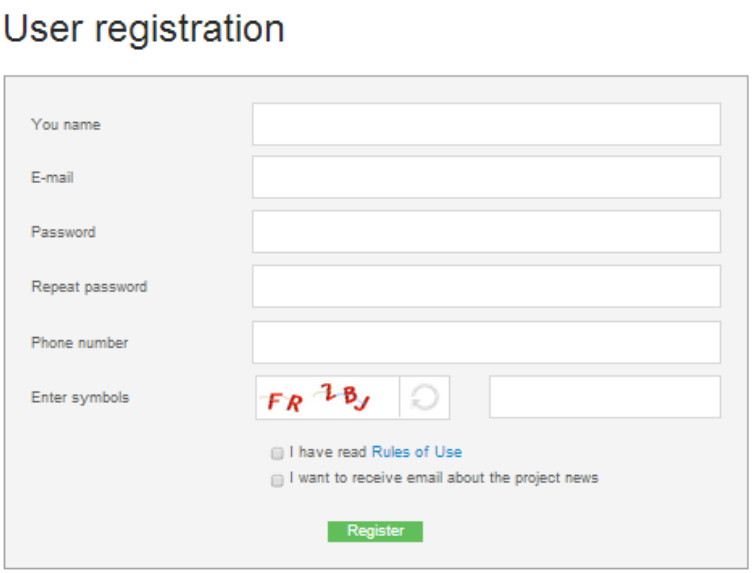

Рис. 4.5 – Реєстраційне вікно

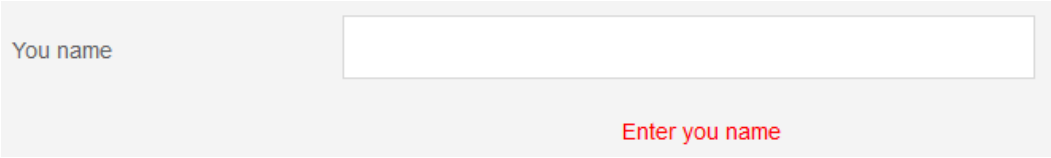

Рис. 4.6 – Попередження про помилку

Як видно з рис. 4.4 головне меню для а аутентифікованого користувача розділене на шести пунктів:

- Main;
- Knowledge Bases;
- Workspace;
- Patterns Analyzer;
- Exit.

Для не аутентифікованого користувача розділів в меню буде тільки три(рис. 4.7):

- Main;
- Register;
- Enter.

Рис. 4.7 – Меню для не аутентифікованого користувача

Пункт «Main» для аутентифікованого і не аутентифікованого користувача одинакові і призначенні для привітального повідомлення(рис. 4.8):

Рис. 4.8 – Відображення «Main»

Пункт «Register» представлений на рисунку 4.5 і призначений для реєстрації нових користувачів. А пункт «Enter» призначений для входу в систему вже зареєстрованих користувачів(рис. 4.9).

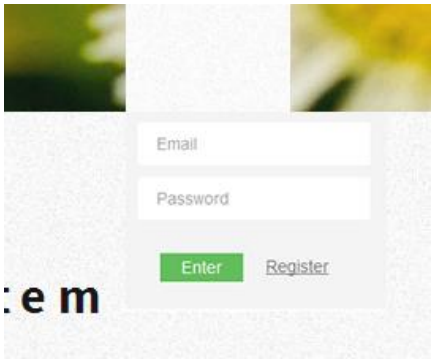

Рис. 4.9 – Форма входу в систему

«Knowledge Bases» надає меню для роботи з основними типами даних, такі як:

- Типи атрибутів шаблонів;
- Атрибути шаблонів;
- Типи атрибутів вимог;
- Атрибути вимог;
- Характеристика;
- Під характеристика;
- Тип збору оцінки.

Редагування проводиться в зручному вікні де користувач з легкістю може ввести нові елементи «Knowledge Bases»(рис. 4.10).

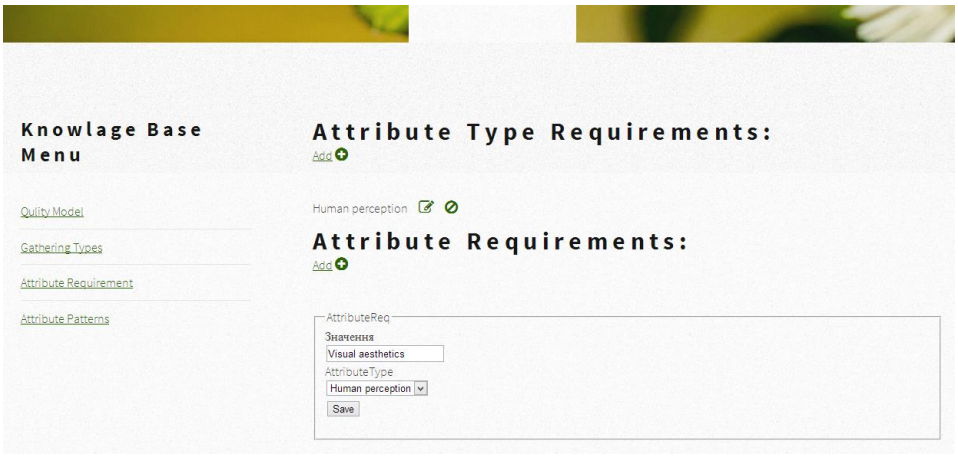

Рис. 4.10 – Приклад вводу елементів в системі

Як і при регістрації у разі помилки користувачу відобразиться відповідне повідомлення з помилкою. В усіх розділах користувачу дозволяється добавляти, редагувати і видаляти елементи системи. Пункт «Workspace» відобразить меню для роботи з проектами. Створивши проект користувач може добавити до нього вимогу або спимсок вимог з описом. А також користувач може добавляти нові шаблони проектування з їх описом. Варто зауважити, що список шаблонів є загальним для всіх користувачів системи і тому додавання нового шаблону одним користувачем дасть можливіст виокристати його і для інших. Створення, редагування, видалення проектів і вимог стилізоване у відповідності до стилю веб-сайту.

Після створення проекту з вимогами і перейшовши в пункт меню «Patterns Analyzer» користувач можемо обрати проект з випадаючого списку і провести аналіз вибору шаблонів (рис. 4.11).

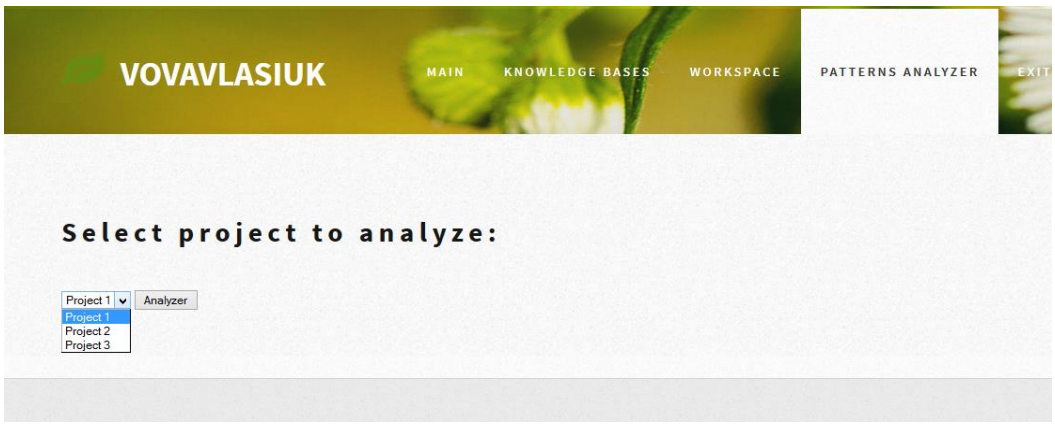

Рис. 4.11 – Відображення пункту «Patterns Analyzer»

Після проведення аналізу вибору архітектурних шабалонів користувачу відображається результати (рис. 4.12).

Рис. 4.12 – Вивід результатів

3.7 Тестування програмної системи

Тестування – невід’ємна складова програмної інженерії та процесу розробки будь-якого програмного засобу. Тестування проводять з метою виявлення дефектів, що не усунулись в процесі розробки, перевірки відповідності програмних засобів висунутим вимогам, покращення якості програмного засобу.

Тестування програмних систем розподіляють на різноманітні категорії за певними показниками: об’єктом досліджень, знанням структури об’єкта тестування, ступенем автоматизації, виглядом представлення результатів.

Одним із базових видів тестування відносно початкового коду та структури програми є модульне тестування. Модульне тестування передбачає розбиття об’єкта тестування (програмної системи в нашому випадку) на певні ізольовані складові, які складають функціональну одиницю – модуль. Парадигма модульного тестування базується на принципі, що коректне функціонування частин забезпечує коректність роботи усієї системи.

Модульне тестування стало поширеним завдяки своїм перевагам:

- Можливості детального тестування ізольованих нетривіальних функцій чи методів
- Підтримка відстеження змін, що можуть призвести до регресії системи
- Забезпечення можливості рефакторингу існуючого коду та повторного тестування
- Розділення концепцій тестування інтерфейсу та реалізації

Найпопулярнішим і найбільш повнофункціональним засобом модульного тестування для програм, що розроблені для виконання платформою .NET є NUnit. NUnit – це середовище модульного тестування програм .NET на основі відкритого коду. Це середовище було портовано із мови Java (бібліотека JUnit). Перші версії засобу NUnit були написані на мові J#, але згодом переписані на C# із підтримкою останніх можливостей середовища виконання .NET.

Крім платформи для тестування засіб NUnit включає візуальні компоненти для відстеження процесу тестування. Компоненти можуть бути окремими прикладними додатками – наприклад Visual NUnit, або вбудованими в середовище розробки, як Visual NUnit 2010 for Visual Studio.

Середовище для модульного тестування NUnit функціонує на базі кількох збірок .NET бібліотек, що створенні для розрізнення помічених для тестування методів та виконання тестів. Для мови C# тести створюються як окремі класи .NET із атрибутами TestFixture, а модульні тести являють собою без параметричні методи із void типом повернення, що помічені атрибутом Test.

Для можливості запуску тестів необхідно включити в проект збірку .NET nunit.framework, та простір імен NUnit. Це дозволить викликати методи для перевірки виконання тестів та маркувати тести відповідними атрибутами. Крім того для успішного виконання тестів цільова збірка, та бібліотека тестування повинні успішно компілюватися – тобто не містити помилок програмного коду етапу компіляції. Структура тесту NUnit зазвичай містить етапи підготовки даних, проведення дії, що тестується та перевірку результату із еталонним значенням. Наприклад:

Лістинг 4.5 – Приклад NUnit тесту

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using NUnit.Framework;
namespace PatternOracleTests
{
    [TestFixture]
    classRequirementTest
    {
        [Test]
        public void Add_Product_In_DB()
        {
            DreamsIT.PatternsOracle.Model.Concret.EFProjectRepository rep =
new DreamsIT.PatternsOracle.Model.Concret. EFProjectRepository ();
            var projLiat= rep.ItemList();
            var project = attributeLiat.First();


DreamsIT.PatternsOracle.Model.Concret.EFRequirementModelRepository pta = new
DreamsIT.PatternsOracle.Model.Concret. EFRequirementModelRepository ();
          var requ = new RequirementModel(){Name = "Test",
                          Project = project,
                            UserId = 1
                       };
            pta.SaveItem(requ);

            var reqTable = pta.ItemList();
            var reqName = productsTable.Last().Name;

            Assert.AreEqual("Test", req);
        }
    }
}
```

Даний програмний код отримує з бази даних відомості про існуючі проекти, створює нову модель вимог із іменем “Test” та виконує вставку цієї моделі в проект

з відповідними відомостями. Потім виконується вибірка моделей вимог та перевірка чи є іменем останньої моделі вимог слово “Test”.

Для виконання тесту в середовищі розробки Visual Studio 2012 необхідно виконати запуск утиліти Visual NUnit 2012, скомпілювати збірку із тестом та програмою, обрати її як цільову для тестів та запустити тест. У разі успішного проходження біля тесту появиться відмітка зеленого кольору, у випадку невдачі – червоного.

Рис. 4.13 - проходження NUnit тестів

Для підтвердження працездатності та коректності функціонування програмного системи вибору архітектурних шаблонів проведено написання модульних тестів для усіх нетривіальних та ресурсоємних методів – таких як вставка, оновлення та видалення записів, зв’язування записів. Успішне проходження модульних тестів засвідчило функціональну повноту, надійність та безпечність використання програмної системи.

3.8 Висновок до розділу

Основні результати даного розділу полягають в наступному:

1. Розроблено БД і БЗ стандартизованих компонентів моделей якості та потреб замовника, які дають змогу реалізувати метод вибору архітектурних патернів.

2. Розроблено інструментальний засіб для підтримки методів побудови моделей вимог до архітектури програмного забезпечення. У даному засобі реалізовано можливість збору потреб замовника і об’єднання їх в групи. Даний засіб надає можливість роботи з БД та полегшує створення вимог до архітектурних патернів ПЗ.

3. Розроблено інструментальний засіб для підтримки методів побудови моделей архітектурних патернів, який надає можливість описувати їх і добавляти до вибору в нейронній мережі. Даний засіб надає можливість роботи з БД та надає інструменти роботи з архітектурними патернами.

4. Розроблено інструментальний засіб для підтримки методу детектування архітектурних шаблонів програмного забезпечення відповідно до заданих вимог. Даний засіб надає можливість роботи з БД та обчислює ймовірність вибору шаблона.

5. Протестовано роботу програмної системи за допомогою NUnit тестів. За рахунок чого доведено ефективність роботи системи.

РОЗДІЛ 4
ОРГАНІЗАЦІЙНО-ЕКОНОМІЧНА ЧАСТИН

Метою дипломної роботи є розробка програмної систему для вибору архітектурних шаблонів ПЗ, зв’язку з поставленими вимогами. Головною метою розділу є встановлення економічної доцільності проведення даної розробки.

4.1 Розрахунок норм часу на виконання науково-дослідної роботи

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

Розробку поділяють на декілька етапів, що дозволить полегшити і структуризувати виконання розробки.

Основні етапи при виконанні розробки веб-сайту наступні:

1. Підготовка опису задачі.
2. Збір необхідної інформації по алгоритмі роботи програми.
3. Збір необхідної інформації по розробці програмного забезпечення.
4. Вибір програмного забезпечення.
5. Розробка структури програмного забезпечення.
6. Розробка дизайну програмного забезпечення.
7. Розробка шаблону.
8. Розробка програмного забезпечення.
9. Тестування роботи програмного забезпечення.

Витрати часу по окремих операціях технологічного процесу відображені в таблиці 4.1.

Виконавцем усіх операцій по розробці веб-сайту являється інженер.

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

Таблиця 4.1

Операції технологічного процесу та час їх виконання

№ п/п	Назва операції (стадії)	Виконавець	Середній час виконання операції, год.
1.	Підготовка опису задачі.	Інженер	12
2.	Збір необхідної інформації по алгоритмі роботи програми.	Інженер	30
3.	Збір необхідної інформації по розробці програмного забезпечення.	Інженер	30
4.	Вибір програмного забезпечення.	Інженер	10
5.	Розробка структури програмного забезпечення.	Інженер	25
6.	Розробка дизайну програмного забезпечення.	Інженер	26
7.	Розробка шаблону.	Інженер	20
8.	Розробка програмного забезпечення.	Інженер	67
9.	Тестування роботи програмного забезпечення	Інженер	20
Разом			**240**

4.2 Визначення витрат на оплату праці та відрахувань на соціальні заходи

Відповідно до Закону України “Про оплату праці” заробітна плата – це “винагорода, обчислена, як правило, у грошовому виразі, яку власник або уповноважений ним орган виплачує працівникові за виконану ним роботу”.

Розмір заробітної плати залежить від складності та умов виконуваної роботи, професійно-ділових якостей працівника, результатів його праці та господарської

діяльності підприємства. Заробітна плата складається з основної та додаткової оплати праці.

Основна заробітна плата нараховується на виконану роботу за тарифними ставками, відрядними розцінками чи посадовими окладами і не залежить від результатів господарської діяльності підприємства.

Додаткова заробітна плата – це складова заробітної плати працівників, до якої включають витрати на оплату праці, не пов'язані з виплатами за фактично відпрацьований час. Нараховують додаткову заробітну плату залежно від досягнутих і запланованих показників, умов виробництва, кваліфікації виконавців. Джерелом додаткової оплати праці є фонд матеріального стимулювання, який створюється за рахунок прибутку.

При розрахунку заробітної плати кількість робочих днів у місяці слід в середньому приймати – 24,5 дні/міс., або ж 196 год./міс. (тривалість робочого дня – 8 год.).

Місячний оклад кожного працівника слід враховувати згідно існуючих на даний час тарифних окладів. Рекомендовані тарифні ставки: керівник проекту – 4,5...8,0 грн./год., інженер – 4,0...6,0 грн./год., консультант – 3,5...6,5 грн./год., технік – 3,0...4,5 грн./год., лаборант – 2,0...3,5 грн./год.

Основна заробітна плата розраховується за формулою:

$$З_{осн.} = Т_с \cdot К_г \text{ ,} \qquad (4.1)$$

де $Т_с$ – тарифна ставка, грн.;

$К_г$ – кількість відпрацьованих годин.

Оскільки всі види робіт в даному випадку виконує інженер, то основна заробітна плата буде розраховуватись тільки за однією формулою:

$$З_{осн.} = 5 \cdot 240 = 1200 \text{ грн.}$$

Додаткова заробітна плата становить 10–15 % від суми основної заробітної плати.

$$З_{дод.} = З_{осн.} \cdot К_{допл.}, \quad (4.2)$$

де $К_{допл}$ – коефіцієнт додаткових виплат працівникам, 0,1–0,15 (візьмемо його рівним 0,15).

$$З_{дод} = 1200 \cdot 0,15 = 180 \text{ грн.}$$

Звідси загальні витрати на оплату праці ($В_{о.п.}$) визначаються за формулою:

$$В_{о.п.} = З_{осн.} + З_{дод.} \quad (4.3)$$

$$В_{о.п.} = 1200 + 180 = 1380 \text{ грн.}$$

Крім того, слід визначити відрахування на соціальні заходи:

1. фонд страхування на випадок безробіття – 1,4 %;
2. фонд по тимчасовій втраті працездатності – 3,1 %;
3. пенсійний фонд – 33,3 %.

У сумі зазначені відрахування становлять 37,8 %.

Отже, сума відрахувань на соціальні заходи буде становити:

$$В_{с.з.} = Ф_{оп} \cdot 0,378 , \quad (4.4)$$

де $Ф_{оп}$ – фонд оплати праці, грн.

$$В_{с.з.} = 1380 \cdot 0,378 = 521,64 \text{ грн.}$$

Проведені розрахунки витрат на оплату праці зведемо у таблицю 4.2.

Таблиця 4.2

Зведені розрахунки витрат на оплату праці

№ п/п	Категорія працівни–ків	Основна заробітна плата, грн.			Додаткова заробітна плата, грн.	Нарахув. на ФОП, грн.	Всього витрати на плату праці, грн. 6=3+4+5
		Тарифна ставка, грн.	К–сть відпра–цьов. год.	Фактично нарах. з/пл., грн.			
А	Б	1	2	3	4	5	6
1.	інженер	5	240	1200	180	521,64	1901,64

Отже, на протязі терміну реалізації проекту по розробці програмного забезпечення для оплати праці необхідно затратити 1901,64.

4.3 Розрахунок матеріальних витрат

Матеріальні витрати визначаються як добуток кількості витрачених матеріалів та їх ціни:

$$M_{ві} = q_i \cdot p_i \,, \qquad (4.5)$$

де: q_i – кількість витраченого матеріалу і–го виду;

p_i – ціна матеріалу і–го виду.

Звідси, загальні матеріальні витрати можна визначити:

$$З_{м.в.} = \sum M_{ві} \,. \qquad (4.6)$$

Проведені розрахунки занесемо у таблицю 4.3.

Таблиця 4.3

Зведені розрахунки матеріальних витрат

Найменування матеріальних ресурсів	Один. виміру	Норма витрат	Ціна за один., грн.	Затрати матер., грн.	Транспортно-заготівель-ні витрати, грн.	Загальна сума витрат на матер., грн.
1	2	3	4	5	6	7
1. Основні матеріали						
Площадка для розміщення веб–сайту	штук	1	4100	–	–	4100
2. Допоміжні витрати						
Використання мережі Internet	години	–	100	100	–	100
Разом:						**4200**

В багатьох випадках існує ряд додаткових витрат які пов’язані із реалізацією проекту, але в більшості випадків їхня сума не перевищує десяти відсотків від загальної собівартості реалізації проекту.

4.4 Розрахунок витрат на електроенергію

Затрати на електроенергію 1–ці обладнання визначаються за формулою:

$$З_в = W \cdot T \cdot S, \quad (4.7)$$

де W – необхідна потужність, кВт;

T – кількість годин роботи обладнання;

S – вартість кіловат-години електроенергії.

Вартість кіловат-години електроенергії слід приймати згідно існуючих на даний час тарифів (0,203 грн. + 20% ПДВ за 1 кВт). Отже, 1 кВт з ПДВ коштує 0,2436 грн.

Потужність комп'ютера для створення проекту – 500 Вт, кількість годин роботи обладнання згідно таблиці 3.1 – 240 години.

Тоді,

$$З_в = 0,5 \cdot 240 \cdot 0,2436 = 29,23 \text{ грн.}$$

4.5 Розрахунок суми амортизаційних відрахувань

Характерною особливістю застосування основних фондів у процесі виробництва є їх відновлення. Для відновлення засобів праці у натуральному виразі необхідне їх відшкодування у вартісній формі, яке здійснюється шляхом амортизації.

Амортизація – це процес перенесення вартості основних фондів на вартість новоствореної продукції з метою їх повного відновлення.

Для визначення амортизаційних відрахувань застосовуємо формулу:

$$A = \frac{Б_В \cdot Н_А}{100\%}, \qquad (4.8)$$

де A – амортизаційні відрахування за звітний період, грн.;

$Б_В$ – балансова вартість групи основних фондів на початок звітного періоду, грн.;

$Н_А$ – норма амортизації, %.

Комп'ютери та оргтехніка належать до четвертої групи основних фондів. Для цієї групи річна норма амортизації дорівнює 60 % (квартальна – 15 %).

Отже, використовуючи в роботі 1 комп'ютер балансовою вартістю 4700 грн. Отже, амортизаційні відрахування будуть рівні:

$$A = 5000 \cdot 5\% / 100\% = 250 \text{ грн.}$$

Оскільки робота виконувалась 240 години, то амортизаційні відрахування будуть становити:

$$A = 250 \cdot 240 / 240 = 250 \text{ грн.}$$

4.6 Обчислення накладних витрат

Накладні витрати пов’язані з обслуговуванням виробництва, утриманням апарату управління спілкою та створення необхідних умов праці.

В залежності від організаційно-правової форми діяльності господарюючого суб’єкта, накладні витрати можуть становити 30–70 % від суми основної та додаткової заробітної плати працівників.

$$H_в = B_{о.п.} \cdot 0{,}3 \ldots 0{,}7 \,, \qquad (4.9)$$

де $H_в$ – накладні витрати.

Отже, накладні витрати:

$$H_в = 1380 \cdot 0{,}3 = 414 \text{ грн.}$$

4.7 Складання кошторису витрат та визначення собівартості НДР

Результати проведених вище розрахунків зведемо у таблицю 4.4.

Таблиця 4.4

Кошторис витрат на НДР

Зміст витрат	Сума, грн.	В % до загальної суми
1	2	3
Витрати на оплату праці (основну і додаткову заробітну плату)	1380	20,3
Відрахування на соціальні заходи	521,64	7,7
Матеріальні витрати	4200	61,8
Витрати на електроенергію	29,23	0,4
Амортизаційні відрахування	250	3,7
Накладні витрати	414	6,1
Собівартість	6794,87	100

Собівартість ($С_в$) програмного продукту розрахуємо за формулою:

$$С_в = В_{о.п.} + В_{с.з.} + З_{м.в.} + З_в + А + Н_в \,. \qquad (4.10)$$

Отже, собівартість програмного продукту дорівнює:

$$С_в = 1380 + 521{,}64 + 4200 + 29{,}23 + 250 + 414 = 6794{,}87 \text{ грн.}$$

4.8 Розрахунок ціни програмного продукту

Ціну НДР можна визначити за формулою:

$$Ц = \frac{С_В \cdot (1 + Р_{рен}) + К \cdot В_{н.і.}}{К} \cdot (1 + ПДВ) \,, \qquad (4.11)$$

де $Р_{рен.}$ – рівень рентабельності, 40 %;

К – кількість замовлень;

$B_{н.і.}$ – вартість носія інформації, грн. (встановлюється лише при розробці програмного продукту);

ПДВ – ставка податку на додану вартість, (30 %).

Оскільки розробка є прикладною, і використовуватиметься тільки для одного підприємства, то для розрахунку ціни не потрібно вказувати коефіцієнти К та Ві.н , оскільки їх в даному випадку не потрібно.

Тоді, формула для обчислення ціни розробки буде мати вигляд:

$$Ц = C_B \cdot (1 + P_{рен}) \cdot (1 + ПДВ). \qquad (4.12)$$

Звідси ціна на проект складе:

$$Ц = 6794{,}87 \cdot (1 + 0{,}4) \cdot (1 + 0{,}3) = 12366{,}66 \text{ грн.}$$

4.9 Визначення економічної ефективності і терміну окупності капітальних вкладень

Ефективність виробництва – це узагальнене і повне відображення кінцевих результатів використання робочої сили, засобів та предметів праці на підприємстві за певний проміжок часу.

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

$$E_p = \frac{П}{C_B}, \qquad (4.13)$$

де *П* – прибуток;

C_B – собівартість.

Плановий прибуток ($П_{пл}$) знаходимо за формулою:

$$П_{пл} = Ц - С_в \, . \qquad (4.14)$$

Розраховуємо плановий прибуток:

$$П_{пл} = 12366{,}66 - 6794{,}87 = 5571{,}79 \text{грн.}$$

Отже, формула для визначення економічної ефективності набуде вигляду:

$$E_р = \frac{П_{пл}}{С_в} . \qquad (4.15)$$

Тоді,

$$E_р = 5571{,}79 / 6794{,}87 = 0{,}82 \, .$$

Поряд із економічною ефективністю розраховують термін окупності капітальних вкладень ($T_р$):

$$T_р = \frac{1}{E_р} \, . \qquad (4.16)$$

Термін окупності дорівнює:

$$T_р = 1 / 0{,}82 = 1{,}2 \text{ роки}$$

Висновок:

В організаційно-економічній частині дипломного проекту було розраховано основні техніко-економічні показники розробки програмної систему для вибору архітектурних шаблонів ПЗ, зв’язку з поставленими вимогами (таблиця 4.5).

Розраховане значення економічної ефективності, яке становить 0,82.

Так само нормальним є термін окупності, який повинен коливатися від 1 до 3 років, тоді розробка вважається доцільною і економічно вигідною. Для даного продукту він становить 1,2 років.

Таблиця 4.5

Техніко–економічні показники НДР

№ п/п	Показник	Значення
1.	Собівартість, грн.	6794,87
2.	Плановий прибуток, грн..	5571,79
3.	Ціна, грн.	12366,66
4.	Економічна ефективність	0,82
5.	Термін окупності, рік	1,2

Отже, даний проект може бути впроваджений та мати подальший розвиток, оскільки він є економічно вигідним за всіма основними техніко-економічними показниками.

РОЗДІЛ 5
ОХОРОНА ПРАЦІ ТА БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

5.1 Охорона праці

Згідно з темою дипломного проекту, розроблено програмну систему для вибору архітектурних шаблонів ПЗ, зв’язку з поставленими вимогами і розробка передбачає використання комп’ютерної техніки.

В Україні розроблені й діють Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин від 10 грудня 1998 р. № 7 (ДСанПіН 3.3.2007-98) та НПАОП 0.00-128-10. У цих нормативних документах регламентується, що приміщення, де люди працюють з ПЕОМ, повинно розміщуватися в північній або північно-східній частині будівлі. Площа одного робочого місця повинна становити щонайменше 6 м^2, об'єм — щонайменше 20 м^3, відстань між робочими столами — щонайменше 2,5 м у ряду і 1,2 м між рядами. Стіни приміщень потрібно фарбувати у пастельні тони з коефіцієнтом відбиття 0,5-0,6[24].

Для того щоб особи, які працюють з ВДТ, меншою мірою втомлювались і зберігали високий рівень працездатності, потрібно раціонально організовувати їхні робочі місця. Зокрема, робоче місце має відповідати основним антропометричним даним людини. Крісло або стілець на робочому місці повинні мати висоту сидіння 40-50 см від рівня підлоги, а також відповідний кут нахилу спинки.

Монітори потрібно розміщувати на висоті рівня очей (висота від підлоги до нижнього краю екрана має становити 95-100 см) на відстані 60-70 см від оператора (відстань від краю столу — 50-70 см). Кут зору працюючого щодо екрана має дорівнювати 10-20°, але не більше 40°, кут між верхнім краєм монітора і рівнем очей користувача має становити менш як 10°. Найдоцільніше

розміщувати екран перпендикулярно до лінії погляду користувача. Кут нахилу екрана по вертикалі має становити 0-30°[24]. З цією метою сучасні монітори комплектують підставкою з поворотним кронштейном, що дає змогу регулювати кут нахилу монітора і горизонтально обертати його навколо вертикальної осі. Висоту екрана від поверхні підлоги регулюють змінюючи висоту робочої поверхні столу. Іноді монітори встановлюють на спеціальні підставки, що уможливлює його переміщення у просторі у вертикальному та горизонтальному напрямах.

З метою зменшення напруження очей потрібно, щоб відстань між краями сусідніх точок зображення на моніторі не перевищувала гранично оптимальний розмір літеро-цифрових знаків — 16-20', складних знаків — 35-40. Оптимальні співвідношення параметрів літер і цифр такі: ширина знака — 0,75 їх висоти, товщина ліній при зворотному контрасті — 1/6-1/8, відстань між знаками — 0,25-0,5 висоти знака, між словами — 0,75-1, між рядками — 0,5-1[24].

Для профілактики загальної втоми і особливо зорового аналізатора важливе значення має організація режиму праці та відпочинку. Загальна тривалість робочого дня не повинна перевищувати 8 год. Частота і тривалість перерв залежать від типу та інтенсивності виконуваних робіт. Під час робіт, які виконуються з великим навантаженням, рекомендуються перерви на 10-15 хв. через кожну годину, а при неінтенсивній і монотонній роботі — на 10-15 хв. через кожні дві години. Кількість мікропауз (тривалістю до хвилини) потрібно регулювати індивідуально. Зміст регламентованих перерв може бути різний: виробнича гімнастика (вправи для очей, гімнастика, спрямована на корекцію вимушеної робочої пози, поліпшення венозного кровообігу, часткову дисфункцію рухової активності), альтернативна допоміжна робота, приймання їжі тощо.

У приміщеннях, де виконуються роботи з ПЕОМ, повинно бути передбачене природне і загальне штучне освітлення. Робочі місця користувачів потрібно розміщувати так, щоб у поле зору не потрапляли вікна і освітлювальні прилади (монітори потрібно розміщувати під кутом 90-105° до вікон і на відстані 2,5-4 м від стін і віконних прорізів). У поле зору користувача не повинні потрапляти поверхні,

що відбивають світло. Покриття столу має бути матовим з коефіцієнтом відбиття 0,25-0,4.

Для штучного освітлення приміщення рекомендується застосовувати світильники матового світла з розсіювачами, а спектральний склад ламп має наближатися до спектру сонячного світла (наприклад, люмінесцентні типу ЛБ). Оптимальна освітленість робочих місць — 400-500 лк.

У разі ураження електричним струмом терміново звільнити потерпілого від дії електричного струму (через відключення електроживлення в кімнаті, загального електроживлення на розподільному щиті або іншим способом). Викликати швидку медичну допомогу (подзвонивши за міським телефоном 103). Надати першу медичну допомогу потерпілому, враховуючи наступне:

- якщо потерпілий знепритомнів, але дихає, його необхідно рівно і зручно вкласти, розстебнути одяг, створити приплив свіжого повітря і забезпечити повний спокій;

- при відсутності ознак життя до прибуття лікарів потерпілому необхідно робити штучне дихання.

Робітники, які працюють, обслуговують, ремонтують ПК несуть відповідальність за порушення вимог. Контролюють виконання даних інструкцій та несуть відповідальність за порушення користувачами ПК правил охорони праці керівники відділів, працівники з охорони праці та пожежної безпеки.

Програмне забезпечення було розроблено відповідно з дотриманням даних нормативними документами.

5.2 Інженерно технічний захист програмно-апаратного забезпечення

Прийняті за останні роки Верховною Радою України закони: “Про Цивільну оборону України”, “Про захист населення і територій від надзвичайних ситуацій техногенного і природного характеру”, чітко визначили призначення і завдання

Цивільної оборони України, відповідальність виконавчої владі всіх рівнів щодо захисту життя і здоров’я людини від наслідків надзвичайних ситуацій.

Оскільки програмно-апаратне забезпечення працює із динними, то потрібно забезпечити захист інформації. Інформаційна безпека має велике значення для забезпечення життєво важливих інтересів будь-якої програмно-технічного засобу. Створення розвиненого і захищеного середовища є неодмінною умовою розвитку ПЗ, в основі якого мають бути найновіші автоматизовані технічні засоби.

Останнім часом в Україні відбуваються якісні зміни у процесах управління на всіх рівнях, які зумовлені інтенсивним упровадженням новітніх інформаційних технологій. Швидке вдосконалення інформатизації, проникнення її в усі сфери життєво важливих інтересів зумовило, крім безперечних переваг, і появу низки стратегічних проблем. Наскільки актуальна проблема захисту інформації від різних загроз, можна побачити на прикладі даних, опублікованих Computer Security Institute (Сан-Франциско, штат Каліфорнія, США), згідно з якими порушення захисту комп'ютерних систем відбувається з таких причин:

- несанкціонований доступ — 2 %
- укорінення вірусів — 3 %;
- технічні відмови апаратури мережі — 20 %;
- цілеспрямовані дії персоналу — 20 %;
- помилки персоналу (недостатній рівень кваліфікації) — 55%.

Таким чином, однією з потенційних загроз для інформації в інформаційних системах слід вважати цілеспрямовані або випадкові деструктивні дії персоналу (людський фактор), оскільки вони становлять 75 % усіх випадків.

Відповідно до вимог законів України "Про інформацію", "Про державну таємницю" та "Про захист інформації в автоматизованих системах" основним об'єктом захисту в інформаційних системах є інформація з обмеженим доступом, передбачену законодавством України, таємницю, конфіденційна інформація, що є державною власністю чи передана державі у володіння, користування, розпорядження.

Загалом, об'єктом захисту в інформаційній системі є інформація з обмеженим доступом, яка циркулює та зберігається у вигляді даних, команд, повідомлень, що мають певну обмеженість і цінність як для її власника, так і для потенційного порушника технічного захисту інформації.

Порушник — користувач, який здійснює несанкціонований доступ до інформації.

Загроза несанкціонованого доступу — це подія, що кваліфікується як факт спроби порушника вчинити несанкціоновані дії стосовно будь-якої частини інформації в інформаційній системі.

Потенційні загрози несанкціонованого доступу до інформації в інформаційних системах поділяють на цілеспрямовані (умисні) та випадкові. Умисні загрози можуть маскуватися під випадкові шляхом довгочасної масованої атаки несанкціонованими запитами або комп'ютерними вірусами.

За відсутності законного користувача, контролю та розмежування доступу до термінала кваліфікований порушник легко використовує його функціональні можливості для несанкціонованого доступу до інформації, що підлягає захисту, шляхом уведення відповідних запитів або команд.

За наявності вільного доступу до приміщення можна візуально спостерігати інформацію на засобах відбиття і документування, викрасти паперовий носій, зняти зайву копію, а також викрасти інші носії з інформацією: лістинги, магнітні носії та інші.

Особливу загрозу становить безконтрольне завантаження програмного забезпечення, в якому можуть бути змінені установки, властивості, дані, алгоритми, введено "троянську" програму або вкорінені комп'ютерний вірус, що виконують деструктивні несанкціоновані дії. Загрозливою є ситуація, коли порушник — санкціонований користувач інформаційної системи, який у зв'язку зі своїми функціональними обов'язками має доступ до однієї частини інформації, а користується іншою за межами своїх повноважень. З боку санкціонованого користувача є багато способів порушення роботи інформаційної системи й

одержання, модифікування, поширювання або знищення інформації, що підлягає захисту. Для цього можна використовувати, насамперед, привілейовані команди введення-виведення, не контрольованість санкціонованості або законності запиту і звернень до баз та банків даних, серверів тощо. Вільний доступ дає порушникові можливість звертатись до чужих файлів і баз даних та змінювати їх випадково або умисно.

Під час технічного обслуговування апаратури можуть бути виявлені залишки інформації на її носіях (поверхні твердих дисків, магнітні стрічки та інші носії). Стирання інформації звичайними методами (засобами операційних систем, спеціальних програмних утиліт) неефективне з погляду технічного захисту інформації. Порушник може поновити і прочитати її залишки, саме тому потрібні тільки спеціальні засоби стирання інформації, що підлягає захисту.

Обробка, передавання та зберігання інформації апаратними засобами інформаційної системи забезпечуються спрацюванням логічних елементів на базі напівпровідникових приладів. Спрацювання логічних елементів зумовлено високочастотним зміщенням рівнів напруг і струмів, що призводить до виникнення в ефірі, ланках живлення та заземлення, а також у паралельно розміщених ланках й індуктивностях сторонньої апаратури електромагнітних полів, які несуть в амплітуді, фазі й частоті своїх коливань ознаки оброблюваної інформації. Використання порушником різних приймачів може призвести до несанкціонованого витоку та перехоплення дуже важливої інформації, що зберігається в інформаційній системі. Зі зменшенням відстані між приймачем порушника й апаратними засобами інформаційної системи ймовірність приймання таких інформаційних сигналів збільшується.

Таким чином, порушники технічного захисту інформації можуть створювати такі потенційні загрози для безпеки інформації в інформаційних системах:

- загрози конфіденційності (несанкціонованого одержання;
- загрози цілісності (несанкціонованої зміни) інформації;

- загрози доступності інформації (несанкціонованого або випадкового обмеження) та ресурсів самої інформаційної системи;
- загрози спостереженості роботи інформаційної системи;
- загрози проникнення комп'ютерних вірусів;
- загрози радіочастотних засобів електромагнітного ураження високопрофесійних порушників.

Загрози порушників технічного захисту інформації можуть здійснюватись:

- технічними каналами: акустичними, оптичними, хімічними тощо;
- каналами спеціального впливу шляхом формування полів і сигналів для руйнування системи захисту або порушення цілісності інформації;
- несанкціонованим доступом шляхом підключення до апаратури та ліній зв'язку, маскування під зареєстрованого користувача, подолання засобів захисту для використання інформації або нав'язування хибної інформації, застосування закладних пристроїв чи програм та вкорінення комп'ютерних вірусів.

В свою чергу, питання ТЗІ розбиваються на два великих класи задач:

- захист інформації від несанкціонованого доступу (НСД)
- захист інформації від витоку технічними каналами.

Під НСД звичайно розуміється доступ до інформації, що порушує встановлену в інформаційній системі політику розмежування доступу. Під технічними каналами розглядаються канали побічних електромагнітних випромінювань і наводок (ПЕМВН), акустичні канали, оптичні канали та інші.

Захист від НСД може здійснюватися в різних складових інформаційної системи:

- прикладне та системне ПЗ.
- апаратна частина серверів та робочих станцій.
- комунікаційне обладнання та канали зв'язку.
- периметр інформаційної системи.

ТЗІ від НСД на прикладному і програмному рівні

Для захисту інформації на рівні прикладного та системного ПЗ нами використовуються:

- системи розмежування доступу до інформації;
- системи ідентифікації та автентифікації;
- системи аудиту та моніторингу;
- системи антивірусного захисту.

ТЗІ від НСД на апаратному рівні

Для захисту інформації на рівні апаратного забезпечення використовуються:

- апаратні ключі;
- системи сигналізації;
- засоби блокування пристроїв та інтерфейс вводу-виводу інформації.

Засоби та заходи ТЗІ

Захист інформації від її витоку технічними каналами зв'язку забезпечується такими засобами та заходами:

- використанням екранованого кабелю та прокладка проводів та кабелів в екранованих конструкціях;
- встановленням на лініях зв'язку високочастотних фільтрів;
- побудовою екранованих приміщень («капсул»);
- використанням екранованого обладнання;
- встановленням активних систем зашумлення;
- створенням контрольованої зони.

Отже, згідно теми дипломного проекту програмно-апаратне забезпечення повинно розгортатись на сумісному обладнанні. Це повинен бути сервер з апаратними характеристиками, які будуть відповідати мінімальним вимогам програмного забезпечення. Потрібно щоб апаратне забезпечення знаходилось у приміщенні яке відповідає потребам компанії, та могло забезпечити належний рівень захисту працівникам, комп’ютерам так і іншому устаткуванню. Дотримання та виконання вимог, наведених в правилах, також здійснення первинних та спеціальних заходів по передбаченню можливих порушень повинно стати

обов’язком для всіх працівників, безпосередньо пов’язаних із навчальним та виробничим процесом

РОЗДІЛ 6
ЕКОЛОГІЯ

6.1 Основні екологічні проблеми

Екологічні проблеми сучасності за своїми масштабами умовно можуть бути розділені на локальні, регіональні і глобальні і вимагають для свого рішення неоднакових засобів і різних за характером наукових розробок.

Приклад локальної екологічної проблеми - завод, що скидає без очищення в річку свої промстоки , шкідливі для здоров'я людей. Органи охорони природи або громадськість повинні через суд оштрафувати такий завод і під загрозою закриття змусити його будувати очисні споруди.

Прикладом регіональних екологічних проблем може служити висихаючи Аральське море з різким погіршенням екологічної обстановки на всій його периферії, або висока радіоактивність ґрунтів в районах, прилеглих до Чорнобиля.

Для вирішення таких проблем вже потрібні наукові дослідження. Наприклад гідрологічні дослідження для вироблення рекомендацій щодо збільшення стоку в Аральське море, у другому випадку - з'ясування впливу на здоров'я населення тривалого впливу слабких доз радіації і розробка методів дезактивації ґрунтів.

Однак антропогенний вплив на природу досягло таких масштабів, що виникли проблеми глобального характеру, про які кілька десятків років тому ніхто навіть не міг підозрювати.

З часу виникнення технічної цивілізації на Землі зведено близько 1/3 площі лісів, пустелі різко прискорили свій рух на зелені зони. Так, пустеля Сахара пересувається на південь зі швидкістю близько 50 км на рік. Катастрофічних розмірів досягло забруднення Океану нафтопродуктами, ядохімікатами, синтетичними миючими засобами, нерозчинними пластиками.

Швидкими темпами відбувається забруднення атмосфери. Поки основним засобом отримання енергії залишається спалювання пального палива, тому з

кожним роком зростає споживання кисню, а на його місце надходять вуглекислота, окисли азоту, окис вуглецю, а так само величезна кількість сажі, пилу і шкідливих аерозолів.

Щорічно у світі спалюється понад 10 млрд. т умовного палива, при цьому викидається в повітря більше 1 млрд. т різних суспензій, серед яких багато канцерогенних речовин. Відповідно до огляду ВНДІ Медичної інформації, за останні 100 років в атмосферу потрапило більше 1,5 млн. т миш'яку, 900 тис т кобальту, 1 млн. т кремнію. Тільки в атмосферу США щорічно викидається понад 200 млн. т шкідливих речовин.

Розпочате в другій половині XX століття різке потепління клімату є достовірним фактом. Середня температура приземного шару повітря в порівнянні з 1956-1957 рр, коли проводився Перший міжнародний геофізичний рік, зросла на 0,7 ^{0}С. На екваторі потепління немає, але чим ближче до полюсів, тим воно помітніше. За Полярним колом воно досягає 2 ^{0}С. На Північному полюсі підлідна вода потеплішала на 1 ^{0}С і крижаний покрив почав підтавати знизу.

Не менш складна екологічна проблема озонового шару. Виснаження озонового шару представляє надзвичайну небезпечну для всього живого на Землі. Озон не допускає небезпечне космічне випромінювання до поверхні Землі. Дослідження причин виснаження озонового шару планети не дали поки остаточних відповідей на всі питання.

Швидке зростання промисловості, що супроводжується глобальним забрудненням природного середовища, небувало гостро поставив проблему сировинних ресурсів. З усіх видів ресурсів першому місці за зростанням потреб на нього і щодо збільшення дефіциту варто віднести прісну воду. 71% всієї поверхні планети зайнята водою, проте прісна вода становить лише 2% загальної кількості, і майже 80 % прісної води знаходяться в льодовому покриві Землі. У більшості промислових районів води вже відчутно не вистачає, і її дефіцит з кожним роком зростає.

Загалом на господарсько-побутові потреби вилучається 10% річкового стоку планети. З них 5,6% витрачаються безповоротно. Якщо безповоротний забір води буде і далі збільшуватися в тому ж темпі, що й тепер (4-5% щорічно), то до 2018 р. людство може вичерпати всі запаси прісних вод в геосфер . Положення ускладнюється тим, що велика кількість природних вод забруднюється промислово-побутовими відходами. Все це в кінцевому рахунку потрапляє в Океан, який і без того піддається сильному забрудненню.

У перспективі тривожно йде справа і з іншим природним ресурсом, що вважався раніше невичерпним - киснем атмосфери . При спалюванні продуктів фотосинтезу минулих епох - горючих копалин , відбувається зв'язування вільного кисню в сполуки. Орієнтовно в надрах Землі міститься $6{,}4\times10^{15}$ т горючих копалин, на спалювання яких було б потрібно $1{,}7\times10^{16}$ т кисню, тобто більше, ніж його налічується в атмосфері.

Досі перспективним джерелом ресурсів зв'язують по інерції з так званими не відновлювальними факторами природнього середовища: запасами залізних руд, кольорових металів, горючих копалин, дорогоцінних каменів, мінеральних солей і тому подібних. Терміни розробки родовищ цих ресурсів свідомо кінцеві і варіюються залежно від багатства змісту їх у земній корі. Вважається, що при нинішніх темпах видобутку запасів свинцю, олова, міді може хватити на 20-30 років. Терміни невеликі, а тому вже заздалегідь шукаються джерела компенсації й економії дефіцитної сировини. Зокрема, вдосконалення методів видобутку дозволяє при ступити до розробки порід з бідним вмістом потрібних елементів і подекуди вже прийнялися за переробку відвалів гірської породи. У перспективі можна буде витягати потрібні елементи в будь-якій кількості з найпоширеніших у природі порід, наприклад з граніту.

По-іншому йде з ресурсами, які здавна звикли вважати поновлюваних і які дійсно були такими доти, поки зрослі темпи їхнього споживання і забруднення середовища не підірвали здатність комплексів до самоочищення і самовідновлення. Причому ці підірвані здібності не поновлюються самі собою, а, навпаки,

прогресивно йдуть на спад у міру нарощування темпів індустрії в колишньому технологічному режимі. Однак свідомість людей все ще не встигло перебудуватися. Воно, як і техніка, працює в колишньому екологічно безтурботному режимі, вважаючи воду, повітря і живу природу невичерпними.

Екологічна проблема поставила людство перед вибором подальшого шляху розвитку: чи бути йому як і раніше орієнтований на безмежне зростання виробництва або це зростання має бути узгоджений з реальними можливостями природного середовища і людського організму, сумарний не тільки з близькими, а й з віддаленими цілями соціального розвитку.

У виникненні та розвитку екологічної кризи особлива, визначальна роль належить технічному прогресу. По суті справи виникнення перших знарядь праці і перших технологій привели до початку антропогенного тиску на природу і виникнення перших спровокованих людиною екологічних катаклізмів. З розвитком техногенної цивілізації відбувалося збільшення ризику екологічних криз і обваження їх наслідків.

6.2 Статистичні показники екологічних явищ

Статистичний показник — це узагальнююча характеристика для кількісного виміру екологічних явищ. Показник — це не тільки одне число, а і його назва. Статистичний показник виражається числом і розмірністю.

Кожен з статистичних показників має три характеристики:

- визначеність, кількість і якість;
- модель розрахунку, екологічний зміст і числове значення змісту;
- адекватність відображення, точність вимірювання і достовірність інформації (рис. 6.1).

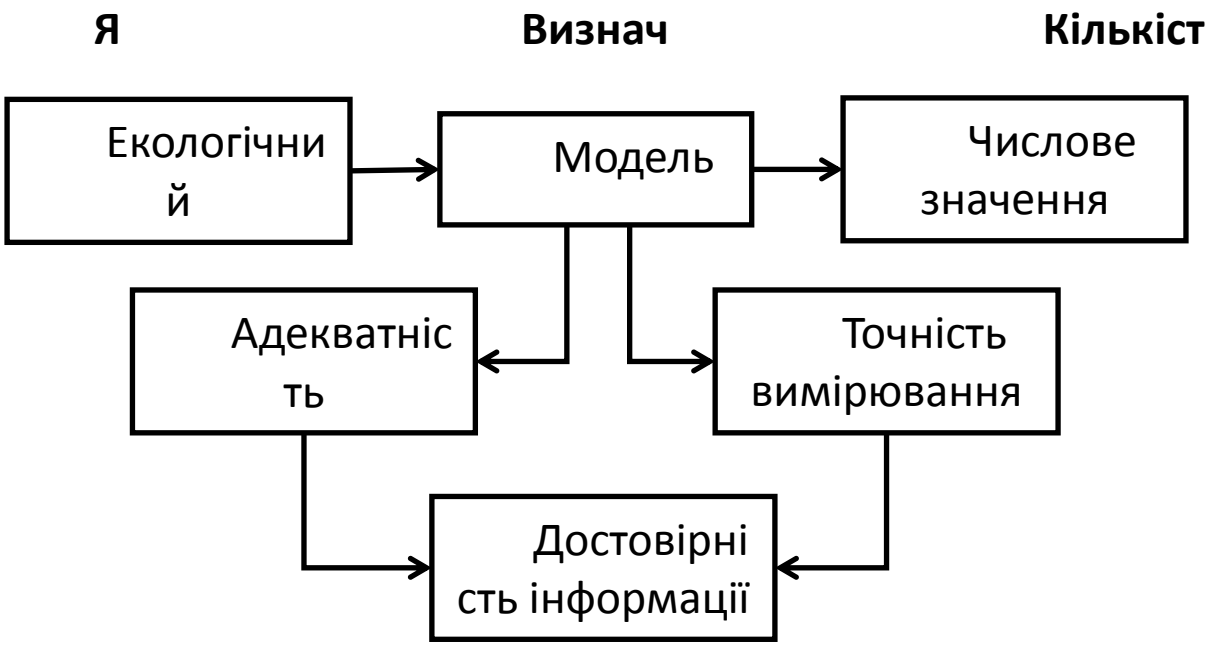

Рис. 6.1 – Статистична модель показників

У статистиці введена класифікація статистичних показників (таблиця 6.1).

Таблиця 6.1

Класифікація видів статистичних показників

За якістю	За кількістю	За характеризованою властивістю
1. Показники властивостей конкретних обсягів	1. Абсолютні	1. Прямі
2. Показники статистичних властивостей будь-яких масових явищ, процесів	2. Відносні	2. Зворотні

За якістю розрізняють показники: властивостей конкретних об'єктів та статистичних властивостей будь-яких масових явищ і процесів. До показників

властивостей конкретних об'єктів належать такі показники: економічні, макроекономічні, демографічні та ін.

Показники статистичних властивостей — це узагальнювальні показники статистики, які використовуються для будь-яких явищ і процесів незалежно від їх конкретного змісту. До показників статистичних властивостей належать: середні значення, середнє квадратичне відхилення, коефіцієнти варіації, структури і характеру розподілу, швидкості і темп зміни, коливання в динаміці та інші.

Відносні статистичні показники дістають через порівняння абсолютних або відносних показників у часі, просторі порівняно з деяким рівнем. Відносні величини можуть бути безрозмірними або відносними показниками. Відносні показники можна поділити на групи, що характеризують:

- структуру об'єкта — відношення частини об'єкта до всього об'єкта. Частка виражається у відсотках або профілях (тисячних). Відносні величини структури дають можливість порівняти склад інгредієнтів об'єкта у різні моменти часу, тобто виявити структурні зрушення;
- динаміку процесу — відношення показника об'єкта в поточний період часу до цього показника об'єкта в попередній період. Такі показники називають темпами зростання. Вони виражаються у разах або відсотках. Темп зростання показує, у скільки разів показник змінився в поточному періоді порівняно з попереднім. До відносних показників динаміки процесу також належать: темп приросту, параметри тренду, коефіцієнти коливання, індексні показники динаміки;
- щільність взаємозв'язку між різними ознаками. між варіацією результативного показника і варіацією факторів. До цих показників належать коефіцієнти еластичності, кореляції, регресії, детермінації;
- співвідношення вторинних ознак одного і того самого об'єкта. Наприклад, коефіцієнт очистки повітря дорівнює відношенню маси видалених шкідливих відходів до об'єму повітря, яке підлягає очищенню;
- відношення фактично спостережуваних величин ознаки до його планових, нормативних, максимально можливих;

- відносні показники, що характеризують відношення ознак різних об'єктів.

Статистичне спостереження - це спланована, науково організована реєстрація масових даних про соціально-економічні й екологічні явища та процеси. Статистичне спостереження може бути первинним або вторинним. Первинне - це реєстрація даних, що надходять безпосередньо від об'єкта, який їх продукує. Вторинне - збирання раніше зареєстрованих та оброблених даних.

Статистичні дані - це масові системні кількісні характеристики соціально-економічних явищ і процесів. Статистичні дані мають відповідати певним вимогам : бути вірогідними; повними; своєчасними; порівнянними за часом або у просторі; доступними.

Отже, статистичні дані в екології є основою для виявлення і обґрунтування емпіричних закономірностей. Без конкретних кількісних даних, що характеризують функціонування досліджуючих еколого-географічних об'єктів, не завжди можна визначити практичне значення застосованої моделі, навіть якщо метою є виявлення переважно якісних закономірностей.

6.3 Система екологічних показників

Система показників охорони навколишнього середовища, яка діє в Україні, ґрунтується на розробленій у 70-ті роки системі статистичної звітності, що стосується навколишнього середовища. Головним завданням цієї галузі стало забезпечення органів управління та планування інформацією, яка була необхідна для визначення стратегії та тактики природокористування і охорони навколишнього середовища в країні, заходів з регулювання впливу господарської діяльності на довкілля.

Система показників стану навколишнього середовища безпосередньо пов'язана з чинною системою статистичної звітності. Створена форма статистичної звітності забезпечила збір даних про найбільш гострі проблеми, пов'язані з антропогенним впливом на окремі складові навколишнього середовища і поклала

основу для побудови системи показників, яка мала виражений по елементний характер і складалася з десяти основних розділів. В більшості розділів можна виділити шість груп показників:

- наявність та склад забруднень;
- показники антропогенного впливу, що викликає ті чи інші зміни навколишнього середовища;
- природоохоронні заходи;
- показники якісного стану або ступеня забруднення (в регіонах і населених пунктах);
- витрати на охорону природи;
- ефективність природоохоронних витрат.

Міжнародне співтовариство на даному етапі розглядає показники стану навколишнього середовища як комплексний інструментарій для виміру та репрезентації еколого-економічних тенденцій у країні.

Чинна в Україні система статистичної звітності в галузі охорони навколишнього середовища не орієнтована на оцінку реакції екосистем на техногенний вплив і критичні параметри впливу для конкретних екосистем та груп населення, а відображає натуральні об'єми забруднювальних речовин і вартісні показники дотримання підприємством чи місцевим органом влади природоохоронного законодавства та планових параметрів проведення природоохоронних заходів. З точки зору економічних показників система статистичної звітності в Україні в галузі охорони навколишнього середовища оперує, в основному, опосередкованими показниками стану навколишнього середовища, тоді як, виходячи з міжнародних вимог, необхідно впроваджувати інтегральні показники прямої дії, що відображали б еколого-економічні процеси на національному рівні.

Охорона навколишнього природного середовища пов'язана з розробленням і здійсненням комплексу екологічно спрямованих заходів, що запобігають або знижують негативний вплив антропогенної діяльності на природу.

Природоохоронні заходи розглядаються у вузькому і широкому розумінні. У вузькому розумінні природоохоронні заходи - це ті види господарської діяльності, які безпосередньо спрямовані на вирішення певних природоохоронних завдань. Такий розподіл обумовлений тим, що природоохоронні заходи вважаються не універсальною, а вузько цільовою сферою діяльності, спрямованою на досягнення вузьких цілей при обмеженості фінансових і матеріальних ресурсів.

У широкому розумінні до середовище захисних заходів можна віднести всі види господарської діяльності, що як прямо, так і побічно сприяють зниженню або ліквідації негативного впливу дій людини на довкілля.

У кінцевому підсумку це обумовлює зменшення ресурсомісткості виробництва одиниці продукції. Інакше кажучи, зменшується питома потреба в зазначених ресурсах. Безпосередніми наслідками цього є відносне зменшення екологічного тиску на стадіях виробництва: зникає (або зменшується) потреба в ресурсі - зникають (або зменшуються) і негативні наслідки його виробництва.

Основними показниками природоохоронної діяльності в Україні слід вважати, з одного боку, обсяги та напрямки фінансування екологічних заходів, з іншого - різні види екологічних платежів і зборів, які не тільки виступають як одне із джерел природоохоронних видатків, але і є дієвим інструментом мотивації природо спрямованої діяльності.

ВИСКНОВКИ

Основні наукові та практичні результати полягають в наступному:

1. Проведено аналіз наукових публікацій та стандартів галузі інженерії програмного забезпечення для визначення сучасного стану інтеграції процесів забезпечення якості ПЗ на стадіях ЖЦ у результаті якого виявлено, що на сучасному етапі розвитку існує ряд проблем пов’язаних з неоднозначністю представлення вимог до архітектури ПЗ, а це негативно позначається на внутрішній якості програмного забезпечення.

2. Проаналізовано характеристики якості ПЗ, які наведені у нормативних документах і використовуються на практиці розробниками. При цьому виявлено, що критерії якості є корпоративними та неуніфікованими.

3. Досліджено та обґрунтовано вибір методу формулювання вимог до архітектури ПЗ, який базується на моделях якості стандарту ISO 9126, що дає змогу забезпечити повноту вимог та адекватність вибору патернів при проектуванні.

4. Розроблено метод і процедуру детектування шаблонів проектування у вигляді теоретико-множинних нотацій, що дало змогу будувати оптимальні архітектури ПЗ.

5. Розроблено і впроваджено програмну систему автоматичного вибору архітектурних патернів, що дало змогу скоротити час на розробку архітектури ПЗ.

6. Проведено обґрунтування економічної доцільності проведення розробки програмної системи автоматичного вибору архітектурних шаблонів.

7. Досліджено законодавчі акти щодо охорони праці при роботі з ВДТ, де присутня інформаційна система і досліджено інженерно технічний захист програмно-апаратного забезпечення в надзвичайних ситуаціях.

8. Проведено дослідження в галузі основних екологічних проблемах, проаналізовано статистичні показники екологічних явищ і системи екологічних показників.

АНОТАЦІЯ

Тема магістерської роботи: «Методи і засоби аналізу вимог та оцінювання якості патернів архітектур програмного забезпечення». // Магістерська робота // Власюк Володимир Володимирович // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп'ютерно-інформаційних систем та програмної інженерії, група СІм-61 // Тернопіль, 2013 // с. – 135, рис. – 26, табл. – 21, аркушів А1 – 9 , додат. –2 , бібліогр. –25.

Ключові слова: АРХІТЕКТУРНІ ПАТЕРНИ, ВИМОГИ, ЯКІСТЬ, БАЗА ДАНИХ, ПРОГРАМНА СИСТЕМА, ДЕТЕКТУВАННЯ ПАТЕРНІВ.

Основними завданнями магістерської роботи є аналіз наукових публікацій та стандартів галузі інженерії програмного забезпечення для визначення сучасного стану інтеграції процесів забезпечення якості ПЗ на стадіях ЖЦ, дослідження та обґрунтування моделі представлення вимог до архітектури ПЗ для забезпечення повноти і адекватності вибору патернів в проектуванні та досягти кінцевої мети – розробити формалізований метод і процедуру детектування шаблонів для забезпечення оптимальної побудови архітектури ПЗ

У першому розділі магістерської роботи досліджено наукові публікації та стандарти в області інформаційних технологій, присвячені задачам оцінювання якості ПЗ. Проаналізовано моделі якості і їх застосуванні на стадіях ЖЦ ПЗ і різноманітні методів проектування архітектури ПЗ. У результаті такого аналізу доведено необхідність створення механізму автоматичного вибору архітектурних шаблонів.

У другому розділі магістерської роботи проведено аналіз методів проектування та формалізації вимог якості до ПЗ, що дало змогу більш повно та адекватно в уніфікованій та стандартизованій формі представляти вимоги до архітектури програмного забезпечення. Також, обґрунтовано і формалізовано метод детектування архітектурних патернів ПЗ, який базується на роботі нейронної

мережі, забезпечує достовірність результатів детектування і як наслідок дають змогу адекватно встановити відповідність архітектурних шаблонів.

У третьому розділі магістерської роботи розроблено програмну систему детектування архітектурних патернів програмного забезпечення і наведено основні функціональні можливості. Також проведено NUnit тестування програмного забезпечення що відображає надійність програмної системи.

У четвертому розділі магістерської роботи обґрунтовано доцільності проведення науково-дослідницької роботі і впровадження програмного комплексу.

В наступних розділах магістерської роботи розглядається питання охорони праці, безпеки життєдіяльності та екології.

ANNOTATION

The theme of master's work is "Methods and means for requirements analysis and quality evaluation patterns of software architectures". // Master work // Vlasiuk Volodymyr Volodymyrovych // Ternopil Ivan Pul'uj National Technical University, department of Computer Information Systems and Program Engineering, group SIm -61 // Ternopil, 2013 // pages – 135, pictures – 26, tables – 21, sheets A1 – 9, applications – 2, bibliography – 25.

Key words: ARCHITECTURAL PATTERN, REQUIREMENTS, QUALITY, DATABASE, SOFTWARE SYSTEM, DETECTING PATTERNS.

The main objectives the master's work is to analyze scientific publications and industry standards of the software engineering to determine the current state of integration of quality assurance processes in software life cycle stages, research and study models of representation requirements for software architecture to ensure the completeness and adequacy of the choice of patterns in the design and achieve the ultimate goal - develop formalized methods and procedures of detection patterns for optimal construction of software architecture.

The first part of the master's work examined scientific publications and standards of the information technology, dedicated to software quality estimation problems. Analyzed the quality models and their applications to life cycle software stages and various software architecture design methods. As a result of this analysis, proved the necessity establishing a mechanism for automatic selection of architectural patterns.

The second part of the master's work analyzes the methods of design and formalization of requirements to software quality, allowing us to more fully and adequately in a uniform and standardized form to submit requirements for software architecture. Also, grounded and formalized method of detecting architectural pattern software, based on the work of the neural network, provides the reliability of detection results and therefore enable adequate to match architectural patterns.

In the third part of the master's work developed software system architectural pattern detection software and shows the main functionality. Also, we have NUnit test software that reflects the reliability of a software system.

The fourth part of the master's works the expediency of carrying out scientific research work and implementation of the software.

In the following parts of the master's work discusses the questions of Health and Hygiene of the Work, Social and Health Education, and Ecology.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Основы инженерии качества программных систем / [Ф. Андон, Г. Коваль., Т. Коротун, Е. Лаврищева, В. Суслов] –К.: Академпериодика – 2007. – 672 с.

2. Андон Ф. Методы инженерии распределенных компьютерных систем / Андон Ф., Лаврищева Е. // Киев, Изд. «Наукова думка», 1997 г.– 228 с.

3. Визначення витрат на створення ПЗ автоматизованих систем / П. Андон, В. Суслов, Т. Коротун, Г. Коваль, О. Слабоспицька – Проблемы программирования. – 1998. - №3. – С. 23 – 34.

4. Волкова С. Дослідження існуючих підходів підвищення якості програмного забезпечення критичного застосування /С. Волкова, О. Трунов – Науково-технічний журнал “Радіоелектронні і комп’ютерні системи” – Харків: Національний аерокосмічний університет ім. М.Є. Жуковського “Харківський авіаційний інститут”. – 2008. – № 6. – С.202-208.

5. ДСТУ 3918-1999 (ISO/IEC 12207:1995) Інформаційні технології. Процеси життєвого циклу програмного забезпечення, К.: Держстандарт України, 2000 – 49 с.

6. Сомервіл І. Инженерия программного обеспечения 6 –издание / І. Сомервіл – Москва–Санкт–Петербург–Киев – 2002 – 623с.

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

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

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

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

11. Характеристика качества программного обеспечения / Б. Боэм, Дж. Браун, Х. Каспар и др. – М.: Мир – 1981. – 206 с.

12. Коваль Г. *Підхід до моделювання якості сімейств програмних систем* / Г. Коваль – Проблеми програмування – №4 – 2009 – С. 49-58.

13. Липаев В. Качество программных средств. Методические рекомендации / Липаев В. – Под ред. А.А. Полякова. – М: Янус-К. – 2002. – 400 с.

14. Липаев В. Методы обеспечения качества крупномасштабных программных систем / В. Липаев – М.: СИНТЕГ. – 2003. – 510 с.

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

16. Letichevsky A. Basic Protocols, Message Sequence Charts, and the Verification of Requerements Specifications / A. Letichevsky, J. Kapitonova, A.Letichevsky Jr., V. Volkov, S. Baranov, V. Kotlyarov // ISSRE 2004, WITUL, Rennes, 4 November 2005.- pp 112 – 142.

17. ISO/IEC 25010. Software Engineering – product quality. Quality model

18. McCall J. Factors in Software Quality. three volumes / J. McCall, P. Richards, G. Walters // NTIS AD-A049-014, AD-A049-015, AD-A049-055, November 1977.

19. Boehm B. Software Risk Management: Principles and Practices / B. Boehm – IEEE Software, January 1991 – pp. 32-41.

20. Чернецки К., Айзенекер У. Порождающее программирование. Методы, инструменты, применение. – Издательский дом «Питер», 2005. – 730 с.

21. ГОСТ 34.601-90 Информационная технология. Комплекс стандартов на автоматизированные системы. Автоматизированные системы. Стадии создания.

22. W.S. McCulloch and W. Pitts, "A logical Calculus of Ideas Immanent in Nervous Activity", Bull. Mathematical Biophysics, Vol. 5, 1943, pp. 115-133.

23. Горбань А. Н. Обобщенная аппроксимационная теорема и вычислительные возможности нейронных сетей // Сибирский журнал вычислительной математики, 1998, т. 1, № 1. — С. 12—24.

24. ДСанПіН 3.3.2007-98 гігієнічні вимоги до організації роботи з візуальними дисплейними терміналами електронно0обчислювальних машин.

25. Erich Gamuna, Richard Helm, Ralph Johnson, John Vlissides // Design Patterns. Elements of Reusable. Object-Oriented Software: 1997 – 431 с.

Додаток А

Акт впровадження

Додаток Б

Фрагменти коду програмної системи детектування архітектурних шаблонів програмного забезпечення

Лістинг Б.1 – Частини контролера при реалізації серверної частини

```
public class AttributeRequirementController : Controller
  {
    private IAttributeReqRepository _attributeReqRepository;
    private IAttributeTypeReqRepository _attributeTypeReqRepository;
    private IRequirementRepository _requirementRepository;

    public AttributeRequirementController(IAttributeReqRepository attributeReqRepository,
IAttributeTypeReqRepository attributeTypeReqRepository, IRequirementRepository requirementRepository)
    {
      _attributeReqRepository = attributeReqRepository;
      _attributeTypeReqRepository = attributeTypeReqRepository;
      _requirementRepository = requirementRepository;
    }
    /// <summary>
    /// Виводить головний екран роботи з типами
    /// </summary>
    /// <returns></returns>
    public ActionResult Index()
    {
      var listAttribute = _attributeReqRepository.ItemList.ToList();
      var listAttributeType = _attributeTypeReqRepository.ItemList.ToList();
      return PartialView("_index", new AttributeTypeRegVisualModel(){AttributeReqs = listAttribute,
AttributeTypeReqs =  listAttributeType});
    }
    /// <summary>
    /// Редагування типу атрибуту
    /// </summary>
    /// <param name="id">Індентифікатор типу атрибута</param>
    /// <returns></returns>
    public ActionResult EditAttributeTypeReg(int id = 0)
    {
      var model = new AttributeTypeReq();
      if (id != 0)
      {
        model = _attributeTypeReqRepository.GetItemByID(id);
      }
      return PartialView("_editAttributeTypeReg", model);
    }
    /// <summary>
    /// Кінець редагування типу атрибуту
    /// </summary>
    /// <param name="model"> Модель типу атрибуту</param>
    /// <returns></returns>
    public ActionResult EndEditAttributeTypeReg(AttributeTypeReq model)
    {
      if (model.ID == 0)
      {
```

```
            _attributeTypeReqRepository.SaveItem(model);
        }
        else
        {
            var oldModel = _attributeTypeReqRepository.GetItemByID(model.ID);
            oldModel.Name = model.Name;
            TryUpdateModel(oldModel);
            _attributeTypeReqRepository.SaveItem(oldModel);
            model = oldModel;
        }
        return PartialView("_endEditAttributeTypeReg", model);
    }
    /// <summary>
    /// Видалення атрибуту
    /// </summary>
    /// <param name="id"> Індентифікатор типу атрибута </param>
    /// <returns></returns>
    public ActionResult DeleteAttributeTypeReg(int id)
    {
        var arrtType = _attributeTypeReqRepository.GetItemByID(id);
        foreach (var attribute in arrtType.Attributes)
        {
            if (attribute.Requirement != null)
            {
                var requirement = _requirementRepository.GetItemByID(attribute.Requirement.ID);
                requirement.AttributeReqs.Remove(attribute);
                TryUpdateModel(requirement);
                _requirementRepository.SaveItem(requirement);
                _attributeReqRepository.DeleteItem(attribute.ID);
            }
        }
        _attributeTypeReqRepository.DeleteItem(id);
        return PartialView("_emptyView");
    }
    /// <summary>
    /// Редагування атрибути
    /// </summary>
    /// <param name="id"> Ідентифікатор атрибуту </param>
    /// <returns></returns>
    public ActionResult EditAttributeReg(int id = 0)
    {
        var model = new AttributeReq();
        if (id != 0)
        {
            model = _attributeReqRepository.GetItemByID(id);
        }
        return PartialView("_editAttributeReg", model);
    }
    /// <summary>
    /// Кінець редагування атрибуту
    /// </summary>
    /// <param name="model"> Модель атрибуту</param>
    /// <param name="idAttributeType"> Індентифікатор типу атрибуту </param>
    /// <returns></returns>
    public ActionResult EndEditAttributeReg(AttributeReq model, int idAttributeType)
    {
        if (model.ID == 0)
```

```
            {
                model.AttributeType = _attributeTypeReqRepository.GetItemByID(idAttributeType);
                _attributeReqRepository.SaveItem(model);
            }
            else
            {
                var oldModel = _attributeReqRepository.GetItemByID(model.ID);
                oldModel.Value = model.Value;
                oldModel.AttributeType = _attributeTypeReqRepository.GetItemByID(idAttributeType);
                TryUpdateModel(oldModel);
                _attributeReqRepository.SaveItem(oldModel);
                model = oldModel;
            }
            return PartialView("_endEditAttributeReg", model);
        }
        /// <summary>
        /// Видалення атрибуту
        /// </summary>
        /// <param name="id"> Індентифікатор атрибуту </param>
        /// <returns></returns>
        public ActionResult DeleteAttributeReg(int id)
        {
            var attribute = _attributeReqRepository.GetItemByID(id);
            if (attribute.Requirement != null)
            {
                var requirement = _requirementRepository.GetItemByID(attribute.Requirement.ID);
                requirement.AttributeReqs.Remove(attribute);
                TryUpdateModel(requirement);
                _requirementRepository.SaveItem(requirement);
            }
            _attributeReqRepository.DeleteItem(attribute.ID);
            return PartialView("_emptyView");
        }
        /// <summary>
        /// Генерує випадаючий список
        /// </summary>
        /// <param name="id"> Ідентифікатор типу атрибута</param>
        /// <returns></returns>
        public ActionResult RenderDropDownListAttrType(int id = 0)
        {
            var listAttributeType = _attributeTypeReqRepository.ItemList.ToList();
            return PartialView("_renderDropDownListAttrType", new DropDownListAttrType(){ AttributeTypeReqs =
listAttributeType, IdAttributeType = id});
        }
    }
```

Лістинг Б.2 – Частина контролеру регістрації користувачів

```
public class AccountController : Controller
  {
    public ActionResult LogOn()
    {
      return View();
    }
    [HttpPost]
    public ActionResult LogOn(LogOnModel model, string returnUrl)
```

```
{
    if (ModelState.IsValid)
    {
        var isValidPassword = FormsAuthentication.Authenticate(model.UserName, model.Password);
        //Membership.ValidateUser(model.UserName, model.Password);

        if (isValidPassword)
        {
            FormsAuthentication.SetAuthCookie(model.UserName, model.RememberMe);
            if (Url.IsLocalUrl(returnUrl) && returnUrl.Length > 1 && returnUrl.StartsWith("/")
                && !returnUrl.StartsWith("//") && !returnUrl.StartsWith("/\\"))
            {
                return Redirect(returnUrl);
            }
            else
            {
                return RedirectToAction("Index", "Home");
            }
        }
        else
        {
            ModelState.AddModelError("", "The user name or password provided is incorrect.");
        }
    }
    // If we got this far, something failed, redisplay form
    return View(model);
}
//
// GET: /Account/LogOff

public ActionResult LogOff()
{
    FormsAuthentication.SignOut();

    return RedirectToAction("Index", "Home");
}
//
// GET: /Account/Register
public ActionResult Register()
{
    return View();
}
//
// POST: /Account/Register
[HttpPost]
public ActionResult Register(RegisterModel model)
{
    if (ModelState.IsValid)
    {
        var user = _serviceFunctions.CreateAdmin(model).UserName;
        try
        {
            FormsAuthentication.SetAuthCookie(user, false);
            return RedirectToAction("Index", "Home");
        }
        catch
        {
```

```
                    ModelState.AddModelError("", "Error create User");
                }
                 //Attempt to register the user
                //MembershipCreateStatus createStatus;
                //Membership.CreateUser(model.UserName, model.Password, model.Email, null, null, true, null, out
createStatus);

                //if (createStatus == MembershipCreateStatus.Success)
                //{
                //    FormsAuthentication.SetAuthCookie(model.UserName, false /* createPersistentCookie */);
                //    return RedirectToAction("Index", "Home");
                //}
                //else
                //{
                //    ModelState.AddModelError("", ErrorCodeToString(createStatus));
                //}
              }
              // If we got this far, something failed, redisplay form
              return View(model);
            }
          }
```

Лістинг Б.3 – Розмітка сторінки «_Layout»

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
   <title>Diploma Vlasiuk V.V. 2013</title>
   <meta name="keywords" content="" />
   <meta name="description" content="" />
   <link                    href="http://fonts.googleapis.com/css?family=Source+Sans+Pro:200,300,400,600,700,900"
rel="stylesheet" />
   <link href="../../Content/css/default.css" rel="stylesheet" type="text/css" media="all" />
   <link href="../../Content/css/fonts.css" rel="stylesheet" type="text/css" media="all" />
   <script src="@Url.Content("~/Scripts/jquery-1.7.1.min.js")" type="text/javascript"></script>
    <script src="@Url.Content("~/Scripts/jquery.validate.js")" type="text/javascript"></script>
   <script src="@Url.Content("~/Scripts/jquery.validate.unobtrusive.js")" type="text/javascript"></script>
   <script src="@Url.Content("~/Scripts/jquery.unobtrusive-ajax.js")" type="text/javascript"></script>
   <script src="@Url.Content("~/Scripts/PatternsOracleScript.js")" type="text/javascript"></script>
</head>
<body>
   <div id="header-wrapper">
      <div id="header-wrapper2">
         <div id="header" class="container">
```

```
                <div id="logo">
                  <h1>
                    <a href="#" class="icon icon-leaf"><span>VovaVlasiuk </span></a>
                  </h1>
                </div>
                <div id="menu">
                  @Html.Action("GetMainMenu", "Visual")
                </div>
              </div>
            </div>
          </div>
          <div id="wrapper">
            <div id="page" class="container">
              @RenderSection("featured", required: false)
              @RenderBody()
            </div>
          </div>
          <div id="portfolio-wrapper">
            <div id="portfolio" class="container">
              <div class="title">
                <h2>Vlasiuk V V</h2>
              </div>
              <p class="description">Patern oracle system. For a diploma work on the topic: Methods and tools for analysis of requirements and quality evaluation patterns of software architectures</p>
            </div>
          </div>
          <div id="copyright" class="container">
            <p>Ternopil city 2013 (c)  | <a href="http://tntu.edu.ua/">Ternopil Ivan Pul'uj National Technical University </a>| <a href="http://kaf-ki.tntu.edu.ua/" rel="nofollow">Computer systems and networks</a>.</p>
          </div>
        </body>
        </html>
```