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

ПОЯСНЮВАЛЬНА ЗАПИСКА

до дипломної роботи

магістр
(освітній рівень)

на тему: **Ранжування атрибутів моделі якості для задач проєктування програмної ахрітектури**

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

(підпис) Гулик А.Я. (прізвище та ініціали)

Керівник (підпис) Харченко О.Г. (прізвище та ініціали)

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

Рецензент (підпис) (прізвище та ініціали)

м. Тернопіль – 2019

АНОТАЦІЯ

Ранжування атрибутів моделі якості для задач проектування програмної архітектури // Дипломна робота ОР "Магістр" // Гулик Арсен Ярославович // Тернопільський національний технічний університет ім. І. Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра комп'ютерних наук // Тернопіль, 2019 // с. – , рис. – ,табл. – , джерел – .

Ключові слова: ЯКІСТЬ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ, ОЦІНЮВАННЯ ЯКОСТІ ПРОГРАМНИХ ПРОДУКТІВ, ПРІОРИТЕТИ ХАРАКТЕРИСТИК ЯКОСТІ, МЕТОД АНАЛІЗУ ІЄРАРХІЙ

В дипломній роботі досліджено одну із складових частин процесу забезпечення якості програмного забезпечення на етапі проектування архітектури, а досліджено процес і запропоновано метод для присвоєння пріоритетів характеристикам якості архітектури під час оцінювання цих архітектур з використанням методу аналізу ієрархій та для встановлення комунікації між характеристиками якості програмної архітектури та характеристиками якості програмного продукту.

ANNOTATION

Ranging of quality model attributes for software architecture design problems // Diploma paper of Master degree level // Hulyk Arsen Yaroslavovych // Ternopil Ivan Puluj National Technical University, Faculty of Computer Information Systems and Software Engineering, Department of Computer Science // Ternopil, 2019 // p. – , Fig. – , Table. – , Refence. – .

Key words: SOFTWARE QUALITY, SOFTWARE PRODUCTS QUALITY ASSESSMENT, QUALITY CHARACTERISTICS PRIORITIES, ANALYTIC HIERARCHY PROCESS

The thesis examines one of the components of the software quality assurance process at the design stage of the architecture and explores the process and suggests a method for assigning priorities to the architecture quality characteristics when evaluating these architectures using the hierarchy analysis method and for establishing communication between the software architecture quality characteristics and characteristics quality of the software product.

ЗМІСТ

ВСТУП

При застосуванні методів експертного оцінювання програмних архітектур задача присвоєння пріоритетів характеристикам якості зустрічається досить часто. Вона має місце в наступних випадках в даній предметній області:

– При побудові матриці парних порівнянь в MAI.

– При побудові матриці комунікації в методі QFD для встановлення множини характеристик якості програмної архітектури.

– При обрахунку інтегральної якості архітектурного рішення через лінійну згортку.

При цьому проектувальник вирішує наступні задачі:

– Виділити множину характеристик.

– Опитати експертів на предмет переваги кожної характеристики якості над іншими.

– Нормалізувати отриману множину значень.

– Відсортувати отримані значення відповідно до отриманих пріоритетів.

Актуальність теми пов'язана з центральним поняттям, яким є поняття якості – під яким, згідно Міжнародному стандарту ISO 8402-2000, надалі розумітимемо сукупність характеристик об'єкту, що визначають його здібності задовольняти встановленим або передбачуваним потребам. В області створення і застосування нових інформаційних технологій вже давно ведуться дослідження, присвячені оцінюванню якості відповідної продукції. Результати вказаних досліджень знаходять своє віддзеркалення у відповідних Міжнародних стандартах і вітчизняних ДСТУ. Результати таких досліджень можуть практично використовуватись для побудови процедур та систем автоматизованого проектування програмних архітектур.

Мета роботи: розробка методики для присвоєння пріоритетів характеристикам якості, котрі беруть участь у порівнянні методом аналізу ієрархій.

Об’єкт, методи та джерела дослідження: процес оцінювання програмних архітектур методом аналізу ієрархій.

Предмет дослідження: характеристики якості програмної архітектури.

Методи дослідження. Для досягнення мети дипломної роботи використовувались:

– методи узагальнення та аналізу – при проведенні огляду технологій проектування архітектури та використовуваних при цьому методах;

– формалізації та математичного моделювання – при побудові математичних моделей для обчислення пріоритетів характеристик якості для методу аналізу ієрархій.

Наукова новизна отриманих результатів. Наукова новизна полягає у вирішенні задачі присвоєння пріоритетів характеристикам якості архітектури ПЗ для методу аналізу ієрархій. При цьому було отримано такі результати:

– запропоновано шаблон опису вимог до нефункціональних характеристик програмної системи;

– описано область застосування пропонованого методу присвоєння пріоритетів характеристикам якості;

– виконано перевірку запропонованого методу для тестового випадку.

Практичне значення отриманих результатів. Розроблені моделі якості програмних архітектур та методи присвоєння пріоритетів характеристикам якості формалізовані, будуються на стандартах. Завдяки цьому дана методика може бути використана для впровадження в склад автоматизованих систем проектування програмного забезпечення.

1 АНАЛІТИЧНИЙ ОГЛЯД ДЖЕРЕЛ ТА АНАЛІЗ І ОБГРУНТУВАННЯ ПРОБЛЕМИ

Сучасні програмні комплекси інформаційно-пошукових систем масового використання, фінансово-банківських та комерційних систем, корпоративних систем управління, веб-сервісів та багатьох інших характеризуються високою функціональною інтегрованістю, розподіленістю, відкритістю, мультимедійним представленням результатів [1]. Тому, ключовими вимогами до таких систем при проектуванні, або виборі готових альтернативних програмних продуктів є вимоги якості надання інформаційних послуг користувачу. Такі вимоги включають широкий спектр властивостей програмних систем (ПС), таких як зручність у використанні, захищеність, ефективність, безпечність, надійність. Однак при проектуванні ПС розробники забезпечують в основному виконання функціональних вимог, а вимоги якості враховують частково і контроль за їх виконанням проводять на завершальній фазі розробки проекту, що не гарантує заявлену якість кінцевого продукту. Основними причинами такої ситуації є недостатнє використання формальних методів при розробці вимог якості, відсутність ефективних процедур їх комунікації, використання не уніфікованих, не стандартизованих характеристик якості, що не забезпечує адекватності та об’єктивності відображення вимог на наступних стадіях життєвого циклу (ЖЦ) [2].

Продуктивним підходом до вирішення перерахованих проблем може бути використання стандарту [3] при розробці вимог та розробка формалізованих процедур їх комунікації на основі використання моделей якості. Сукупність формалізованих моделей вимог якості та ефективних алгоритмів їх комунікації складає основу для розробки інструментальних засобів автоматизації процесів управління якістю ПС в ЖЦ.

1.1 Методика побудови архітектури та дизайну

Ітеративна техніка, яка може використовуватися при продумуванні і створення прототипу майбутньої архітектури. Вона допоможе звести воєдино ключові рішення, обговорювані в цьому посібнику, включаючи рішення за параметрами якості, архітектурним стилям, типами додатків, технологіям і сценаріями розгортання.

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

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

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

1.1.1 Вихідні дані, вихідні дані і етапи проектування

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

та інші), технологічні вимоги, цільова середу розгортання й інші обмеження.

У ході процесу розробки створюється список значущих з погляду архітектури варіантів використання, аспектів архітектури, які потребують спеціального уваги, і можливих архітектурних рішень, які задовольняють вимогам і обмеженням, виявленим у процесі проектування. Загальною технікою поступової доопрацювання дизайну до тих пір, поки він не буде задовольняти всім вимогам і обмеженням, є ітеративна методика, що включає п'ять основних етапів. Цими етапами є:

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

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

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

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

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

Такий процес створення архітектури припускає ітеративний і інкрементний підхід. Спочатку створюється можливий варіант архітектури – узагальнений дизайн, який може тестуватися з основними сценаріями, вимогам,

відомим обмеженням, параметрам якості і Архітектурної Базі. В ході доопрацювання варіанта архітектури, виявляються додаткові деталі і відомості про дизайн, результатом чого стає розширення основних сценаріїв, коректування загального подання додатка і підходу до вирішення проблем.

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

1.1.2 Визначення цілей архітектури

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

- Початкова визначення завдань архітектури. Від цих завдань буде залежати час, що витрачається на кожну фазу проектування архітектури. Необхідно вирішити, що ви робите: створюєте прототип, проводите тестування можливих варіантів реалізації або виконуєте тривалий процес розробки архітектури для нового додатка.
- Визначення споживачів архітектури. Визначте, чи буде розробляється конструкція використовуватися іншими архітекторами, або вона призначається для розробників і тестувальників, IT-спеціалістів та керівників. Врахуйте потреби і підготовленість цільової аудиторії, щоб зробити розроблювану конструкцію максимально зручною для них.
- Визначення обмежень. Вивчіть всі опції і обмеження застосовуваної технології, обмеження використання та розгортання. Повністю розберіться з усіма обмеженнями на початку роботи, щоб не витрачати час або не стикатися з сюрпризами в процесі розробки програми.

1.1.3 Аналіз архітектури

Аналіз архітектури додатку[3,5,7] – критично важливе завдання, оскільки дозволяє скоротити витрати на виправлення помилок, якомога раніше виявити і виправити можливі проблеми. Аналіз архітектури слід виконувати часто: по завершенні основних етапів проекту і у відповідь на істотні зміни в архітектурі. Створюйте архітектуру, пам'ятаючи обосновних питаннях задаються при такому аналізі, це дозволить як поліпшити архітектуру, так і скоротити час, що витрачається на кожен аналіз.

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

1.1.4 Ключові сценарії

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

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

1.1.5 Важливі з точки зору архітектури варіанти використання

Важливі з точки зору архітектури варіанти використання впливають на багато аспектів дизайну. Вони відіграють особливо важливу роль у забезпеченні майбутнього успіху створюваного додатка. Ці варіанти використання важливі для приймання розгорнутого додатки і повинні охоплювати досить велику частину дизайну, щоб бути корисними при оцінці архітектури. До важливих з точки зору архітектури варіантам використання належать:

– Бізнес-критичний (Business Critical). Варіант використання, що має високий рівень використання або особливу важливість для користувачів або інших зацікавлених сторін, в порівнянні з іншими функціями, або передбачає високий ризик.

– Хто має великий вплив (High Impact). Варіант використання охоплює і функціональність, і параметри якості, або представляє наскрізну функцію, що має глобальний вплив на шари і рівні додатку. Прикладами можуть служити особливо вразливі з точки зору безпеки операції Create, Read, Update, Delete (CRUD).

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

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

1.1.6 Загальне уявлення додатка

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

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

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

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

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

Архітектурний стиль покращує секціонування і сприяє можливості повторного використання дизайну завдяки наданню рішень часто зустрічаються проблем. Типовими архітектурними стилями є сервісно-орієнтована архітектура (Service Oriented Architecture, SOA), клієнт / сервер, багатошарова, шина повідомлень і проектування на основі предметної області. Додатки часто використовують поєднання стилів.

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

1.1.7 Відповідні технології проектування архітектури

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

- Мобільні додатки. Для розробки програми для мобільних пристроїв можуть використовуватися технології шару уявлення, такі як .NET Compact Framework, ASP.NET для мобільних пристроїв і Silverlight для мобільних пристроїв.
- Насичені клієнтські програми. Для розробки додатків з насиченими UI, розгортаються та виконуваними на клієнті, можуть використовуватися поєднання технологій шару уявлення Windows Presentation Foundation (WPF), Windows Forms і XAML Browser Application (XBAP).
- Насичені клієнтські Інтернет-додатки (RIA). Для розгортання насичених UI в рамках Веб-браузера можуть використовуватися модуль Silverlight ™ або Silverlight в поєднанні з AJAX.
- Веб-додатки. Для створення Веб-додатків можуть застосовуватися ASP.NET WebForms, AJAX, Silverlight, ASP.NET MVC і ASP.NET Dynamic Data.
- Сервісні програми. Для створення сервісів, що надають функціональність зовнішнім споживачам систем і сервісів, можуть використовуватися Windows Communication Foundation (WCF) і ASP.NET Web services (ASMX).

1.1.8 Наскрізна функціональність

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

- Аутентифікація і авторизація. Як правильно вибрати стратегію аутентифікації та авторизації, передачі ідентифікаційних даних між шарами і рівнями і зберігання посвідчень користувачів.
- Кешування. Як правильно вибрати техніку кешування, визначити дані, що підлягають кешуванню, де кешувати дані і як вибрати відповідну політику закінчення терміну дії.
- Зв'язок. Як правильно вибрати протоколи для зв'язку між шарами і рівнями, забезпечення слабкого зв'язування між шарами, здійснення асинхронного обміну даними та передачі конфіденційних даних.
- Управління конфігурацією. Як виявити дані, які повинні бути налаштованим, де і як зберігати дані конфігурації, як захищати конфіденційні дані конфігурації і як обробляти їх в серверній фермі або кластері.
- Управління винятками. Як обробляти і протоколювати виключення і забезпечувати повідомлення у випадку необхідності.
- Протоколювання і інструментіроваріе. Як вибрати дані, що підлягають протоколюванню, як зробити протоколювання налаштованим, і як визначити необхідний рівень інструментірованія.
- Валідація. Як визначити, де і як проводити валідацію; як вибрати методики для перевірки довжини, діапазону, формату і типу; як запобігти і відхилити введення неприпустимих значень; як очистити потенційно зловмисний і небезпечний введення; як визначити і повторно використовувати логіку валідації на різних шарах і рівнях додатки.

1.1.9 Базова архітектура і можливі варіанти архітектури

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

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

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

Ітеративний і інкрементний підхід дозволяє позбутися великих ризиків спочатку, итеративно формувати архітектуру і через тестування підтверджувати, що кожна нова базова архітектура є поліпшенням попередньої. Наступні питання допоможуть протестувати новий варіант архітектури, отриманий на підставі «пілота» архітектури:

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

1.1.10 Пілотні архітектури

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

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

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

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

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

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

1.2 Рекомендації по проектуванню компонентів

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

Загальні рекомендації з проектування компонентів.

Розглянемо загальні рекомендації проектування компонентів додатків:

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

- **Принцип єдиності відповідальності (Single responsibility). Клас повинен відповідати тільки за один аспект.**
- **Принцип відкритості / закритості (Open / closed principle). Класи повинні бути розширюваними без необхідності доопрацювання.**
- **Принцип заміщення Лискова (Liskov substitution principle). Підтипи і базові типи повинні бути взаємозамінні.**
- **Принцип відділення інтерфейсу (Interface segregation principle). Інтерфейси класів повинні бути клієнт-специфічними і вузьконаправленими. Класи повинні**

надавати різні інтерфейси для клієнтів, що мають різні вимоги до інтерфейсів.

- **Принцип інверсії залежностей (Dependency inversion principle).**

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

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

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

– Компонент не повинен залежати від внутрішніх деталей інших компонентів.

Кожен компонент або об'єкт повинен викликати метод іншого об'єкта або компонента, і цей метод повинен знати, як обробляти запит і, якщо необхідно, як направити його до відповідних підкомпонентів або інших компонентів. Такий підхід дозволяє створювати більш адаптуються і зручні в обслуговуванні додатка.

– Продумайте, як компоненти будуть взаємодіяти один з одним.

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

– Не змішуйте код наскрізний функціональності і прикладну логіку додатку.

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

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

1.3 Розподіл компонентів по шарах

Кожен шар додатки містить набори компонентів, що реалізують функціональність даного шару. Ці компоненти повинні бути зв'язковими і слабо пов'язаними, щоб забезпечити можливість повторного використання і спростити обслуговування.

1.3.1 Компоненти шару представлення

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

– Компоненти для інтерфейсу користувача.

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

У більшості випадків їх роль полягає в наданні користувачеві інтерфейсу, який забезпечує найбільш відповідне подання даних і логіки додатка, а також в

інтерпретації користувальницької введення і передачі його в компоненти логіки уявлення, які визначають вплив введення на дані і стан програми.

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

– Компоненти логіки представлення.

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

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

– Компоненти Presenter, Controller, Presentation Model і ViewModel.

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

– Компоненти сутностей уявлення.

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

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

1.3.2 Компоненти шару сервісів

Додаток може надавати шар сервісів для взаємодії з клієнтами або використання іншими системами. Компоненти шару сервісів забезпечують іншим клієнтам і додаткам спосіб доступу до бізнес-логікою додатки і використовують функціональність програми шляхом обміну повідомленнями по каналу зв'язку. Зазвичай в шарі сервісів розташовуються наступні типи компонентів:

- Інтерфейси сервісів.

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

- Типи повідомлень.

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

1.3.3 Компоненти бізнес-шару

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

Бізнес-шар зазвичай включає наступні типи компонентів:

–Фасад додатку.

Цей необов'язковий компонент зазвичай забезпечує спрощений інтерфейс для компонентів бізнес-логіки часто шляхом об'єднання безлічі бізнес-операцій в одну, що спрощує використання бізнес-логіки й скорочує кількість залежностей, оскільки зовнішнім зухвалим сторонам немає необхідності знати деталі бізнес-компонентів і відносини між ними.

–Компоненти бізнес-логіки.

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

–Компоненти робочого процесу.

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

–Компоненти бізнес-сутностей.

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

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

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

1.3.4 Компоненти шару доступу до даних

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

–Компоненти доступу до даних.

Ці компоненти абстрагують логіку, необхідну для доступу до базових сховищ даних. Для більшості завдань доступу до даних необхідна загальна логіка, яка може бути виділена і реалізована в окремих допоміжних компонентах,

доступних для повторного використання, або підходящої допоміжної інфраструктурі. Це може спростити компоненти доступу до даних і централізувати логіку, що полегшує обслуговування. Решта завдань, загальні для компонентів шару даних і не відносяться ні до одного набору компонентів, можуть бути реалізовані як окремі службові компоненти. Допоміжні та службові компоненти часто об'єднуються в бібліотеку або інфраструктуру, що полегшує їх повторне використання в інших додатках.

–Агенти сервісів.

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

1.3.5 Компоненти наскрізної функціональності

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

- Компоненти для реалізації безпеки. Сюди відносяться компоненти, що здійснюють аутентифікацію, авторизацію і валідацію.
- Компоненти для реалізації завдань операційного управління. Сюди відносяться компоненти, що реалізують політики обробки виключень, протоколювання, лічильники продуктивності, конфігурацію і трасування.
- Компоненти для реалізації взаємодії. Сюди відносяться компоненти, які взаємодіють з іншими сервісами та додатками.

2 ОБГРУНТУВАННЯ МЕТОЛІВ ТА ЗАСОБІВ ВИРІШЕНЯН ПРОБЛЕМИ

Розробка ахрітектури програних систем є комплексом заходів розробки. Вона включає прийняття рішень про число взаємозалежних конструкторських рішень, що відносяться до області розробки. Кожне рішення потребує вибору з деякої кількості альтренатив, кожна з яких по різному впливає на різноманітні атрибути якості. Крім того, як правило в процесі прийняття рішення бере участь деяка кількість замовників, які мають різні, часто конфліктуючі цілі при досягненні якості, а також існують обмеження проєкту, такі як кошторис чи часові межі.

Для полегшення процесу розробки ахрітектури пропонується кількісний підхід на базі моделі якості, який намагається знайти найкраще можливе співвідношення між конфліктуючими цілями якості у замовників, конкуруючими складовими ахрітектури та обмеженнями проєкту. Підхід використовує методику оптимізації для рекомендації оптимального варіанту ахрітектури. Придатність до застосування пропонованого підходу оцінено на прикладі реальної системи.

Розробка ахрітектури програмного забезпечення (програмної ахрітектури – ПА) для розподілених застосувань є широким колом важливих комплексних заходів. Її важливість базується на тому факті, що вона включає декілька рішень, які буде затратно та важко змінити на наступних етапах, коли виявиться що ці рішення хибні. Розробка також є комплексною, тому що розробник повинен здійснити цілий ряд компромісів для задоволення взаємовиключних ахрітектурних вимог [1].

Наступні складнощі виникають через те, що процес розробки ахрітектури виконується на ранніх етапах життєвого циклу. Через те, що на цьому етапі або немає зовсім ніяких продуктів, або лише декілька, а тому важко, часто неможливо повністю обґрунтувати послідовність рішень розробки. Це особливо стосується рішень розробки, що містять готові компоненти розподіленої

ахрітектури (J2EE, .NET, CORBA), які часто використовуються в ахрітектурі. З цих причин архітектори використовують прототипи та попередній досвід для оцінювання їх розробок.

Додатково в процес розробки часто залучені замовники, кожен з котрих має власні конфліктуючі вимоги до якості. Крім того, розробка програмного забезпечення часто обмежена вартістю проєкту та часовим графіком, котрі завжди мають бути задоволені.

У спробі допомогти полегшити складність розробки ахрітектури це дослідження має за мету надати методику та інструменти для підтримки формалізованого обґрунтування під час розробки ахрітектури.

Ця робота мотивована потребою:

– Допомогти різним замовникам системи виразити бажані цілі якості у вимірюваній формі та формалізувати процес надання пріоритетів цим цілям.

– Допомогти архітекторам визначити ахрітектуру-кандидата, яка найкраще задовольняє цілям якості від замовників та вкладається в обмеження проєкту.

В цьому розділі описується процес розробки на базі моделі якості, який використовує контрольований вибір рішень та їх обґрунтування під час розробки ахрітектури. Новизна даного підходу полягає у використанні методики оптимізації, частково цілочисельного програмування [2], для оптимізації розробки ПА, що включає декілька взаємозалежних рішень розробки.

Даний підхід має переваги порівняно з попередніми розробками, які оцінюють та вибирають між заданими крупноелементними ахрітектурами [3,4] без надання вказівок як досягти цієї ахрітектури. Ми доказуємо, що оцінювання всіх ахрітектур-претендентів є важким, часто неможливим завдання через велику кількість цих альтренатив. Запропонований підхід таким чином оцінює та вибирає між ахрітектурами-кандидатами на дрібноелементному рівні, допомагаючи таким чином замовникам прийти до підходящого рішення.

Тут оцінюється придатність до застосовування пропонованого підходу з застосовуванням вивчення реального випадку впровадженої системи на прикладі

з літературних джерел, що має декілька замовників з різними цілями якості, та множинними взаємозалежними рішеннями розробки та обмеженнями проєкту.

2.1 Обґрунтування методу

Якість програмного забезпечення – це ступінь того, як застосування володіє бажаною комбінацією атрибутів якості [5]. ПА відіграє центральну роль у досягненні системою атрибутів якості. Розробка ахрітектури розподіленого застосування є, однак, за своєю суттю більш складним, ніж для централізованих систем. Розподілені системи повинні мати справу, наприклад, з додатковими моделями відмов, недетермінізмом, вбудованому конфігуруванні та управлінні, масштабованістю машин та мереж. Отже, в результаті необхідно розглядати вимоги якості на ранніх етапах проєктування ПА.

2.1.1 Обговорення якості під час розробки ПА

Спільнота програмної інженерії розробила різні методи для підтримки систематичного обґрунтування різних атрибутів якості (таких як режим реального часу [6], надійність [7] та показники роботи [8]) під час створення ахрітектури застосування. Однак, ці методи охоплюють певні атрибути якості ізольовано. В дійсності атрибути якості взаємодіють один з одним. Наприклад, існує конфлікт між конфігурованістю та показниками роботи [9]; показники роботи впливають також на модифікованість, і кожен атрибут якості також впливає на вартість [10].

Деякі дослідники винайшли методи, що роблять атрибути якості центральними при розробці застосування. Бош [11] пропонує метод, який явно розглядає атрибути якості під час процесу розробки. Хофмейстер та ін. [12] описують середовище, відоме як глобальний аналіз для ідентифікації, пристосування та опису факторів, важливих для ахрітектури, включаючи атрибути якості, на ранніх етапах розробки.

Робота [15] надає середовище, яке розглядає, як кожне рішення розробки діє на простір атрибутів якості. Однак, воно не забезпечує підтримку чіткого виконання аналізу компромісів між конкуруючими рішеннями розробки.

Басс та ін. запропонували метод, орієнтований на атрибути (АДД) [14] для допомоги архітекторам будувати процес розробки на бажаних атрибутах якості. АДД розроблений на базі Атрибуто-Базованих Ахрітектурних стилях (АБАС) [13] та ахрітектурних поданнях (представленнях) [14, 15]. Цей метод надає середовище для прийняття конструкторських рішень з відомими впливами на бажані атрибути якості. Однак, систематизовані знання чи досвід архітектора можуть презентувати більше, ніж одну альтренативу розробки для кожного рішення. У цій ситуації чисельне середовище обґрунтування багатокритеріального вибору може доповнити методи, подібні до АДД.

2.1.2 Кількісне обґрунтування рішення з розробки

Кацман та ін. [4] пропонують метод аналізу вартісних вигод Cost Benefit Analysis Method (CBAM) для числових рішень в термінах аналізу вартісних вигод. Вони використовують кількісний підхід для оцінки та вибору між альтренативними ахрітектурними стратегіями. Ідея полягає в тому, що замовники вибиратимуть стратегії для максимізації повернених інвестицій (ROI). CBAM, однак, не допомагає ідентифікувати ахрітектурні кандидати, оскільки рішення розглядаються незалежно.

Швидше, цей метод базується на методах, подібних до методу аналізу ахрітектурних компромісів (ATAM) [16] для ідентифікації конкуруючих ахрітектурних стратегій. ATAM – це метод оцінки ахрітектури, який сам потребує ахрітектуру як вхідне значення для процесу оцінки.

Мікаел та ін. [9] розробили числовий підхід до порівняння конкуруючих ахрітектур з застосовуванням методу аналізу ієрархій (MAI). Цей метод пропонує структурований шлях до виявлення преференцій атрибутів якості від

замовників та допомагає їм отримати кількісне розуміння вигод та пасивів у різних конкуруючих ахрітектурах.

Мета в [9] та ж сама, що і в цій роботі, але підхід відрізняється. Подібно до CBAM, цей метод передбачає, що створено малий набір конкуруючих ахрітектур, але не дає ніяких вказівок, як досягти цих альтренативних ахрітектур. На відміну існуючим підходам, які оцінюють ахрітектуру на рівні крупних елементів, наш підхід пропонує вказівки на ранніх етапах процесу створення ахрітектури. Це досягається через оцінювання різних дрібних елементів та параметрів розробки, які разом утворюють результуючу ПА.

2.2 Пропонований підхід

2.2.1 Цілі підходу

Метою пропонованого підходу є зменшення складності та збільшення надійності розробки ПА. Для того, що б це зробити, ми систематизуємо процес створення ахрітектури, як це запропоновано раніше у [6]. Це мало б допомогти архітекторам систематично визначати оптимальні комбінації розроблюваних альтренатив (тобто найкращу ахрітектуру). Для досягнення цієї мети даний підхід повинен задовольняти двом головним вимогам:

1. Локальна вимога: для кожного окремого рішення підхід має вибирати альтренативу, котра найкраще задовольняє преференції замовників стосовно атрибутів якості.

2. Глобальна вимога: вибір однієї чи комбінації рішень для окремої альтренативи не повинен порушувати обмеження проєкту, та повинен також підтримувати взаємозалежності рішень.

Для першої вимоги нам потрібно застосовувати механізм, за допомогою якого можна обрахувати значення для кожної альтренативи у відповідності для кожного рішення. Вище значення для однієї альтренативи означає, що вона є кращою у порівнянні з іншою. Таким чином пораховані значення повинні враховувати преференції замовників стосовно атрибутів якості.

Однак, для окремого рішення може не виявитись альтренативи з найвищим значенням, яке б не порушувало обмеження проєкту. Наприклад, вибір окремої альтренативи може спричинити додаткові витрати. Цей приклад показує, що глобальна вимога також має бути задоволена.

Підхід до розробки також має справу з взаємозалежностями між різними рішеннями. Початково ми ідентифікували два типи залежностей (базовані на альтренативах та базовані на контексті залежності), котрі обговорюються далі в цьому розділі. В якості прикладу вважатимемо, що є два рішення по розробці, кожне з яких має декілька альтренатив. Якщо найкраща альтренатива (що дає найвище значення при оцінюванні) для першого рішення конфліктує з найкращою альтренативою другого рішення, то для архітектора важко розв’язати це протиріччя.

Базуючись на огляді, ми стверджуємо , що задача розробки ахрітектури ПС може бути сформульована як задача глобальної оптимізації, оскільки рішення з розробки сильно залежать одне від одного, та вибір будь-якої альтренативи не повинно порушувати глобальних обмежень. Таким чином, ми формулюємо задачу оптимізацію наступним чином: "ми домагаємся максимізувати величину якості ахрітектури ПС для всіх замовників, залучених до вибору ахрітектури, що дає найвищі результати, дотримуючись одночасно забезпечення збереження залежностей та глобальних обмежень".

Наскільки добре відомо, жоден з існуючих підходів до розробки ахрітектури не досліджує потенціал застосування методик оптимізації для розв’язку задач комплексної розробки ПС з застосовуванням множинних взаємозалежних ахрітектурних рішень.

2.2.2 Підхід до оцінювання ПА

Пропонується методика, яка охоплює три кроки, як показано на рис. 2.1. Він починається з першого рішення по розробці та обчислює величини для його потенційних альтренативних рішень.

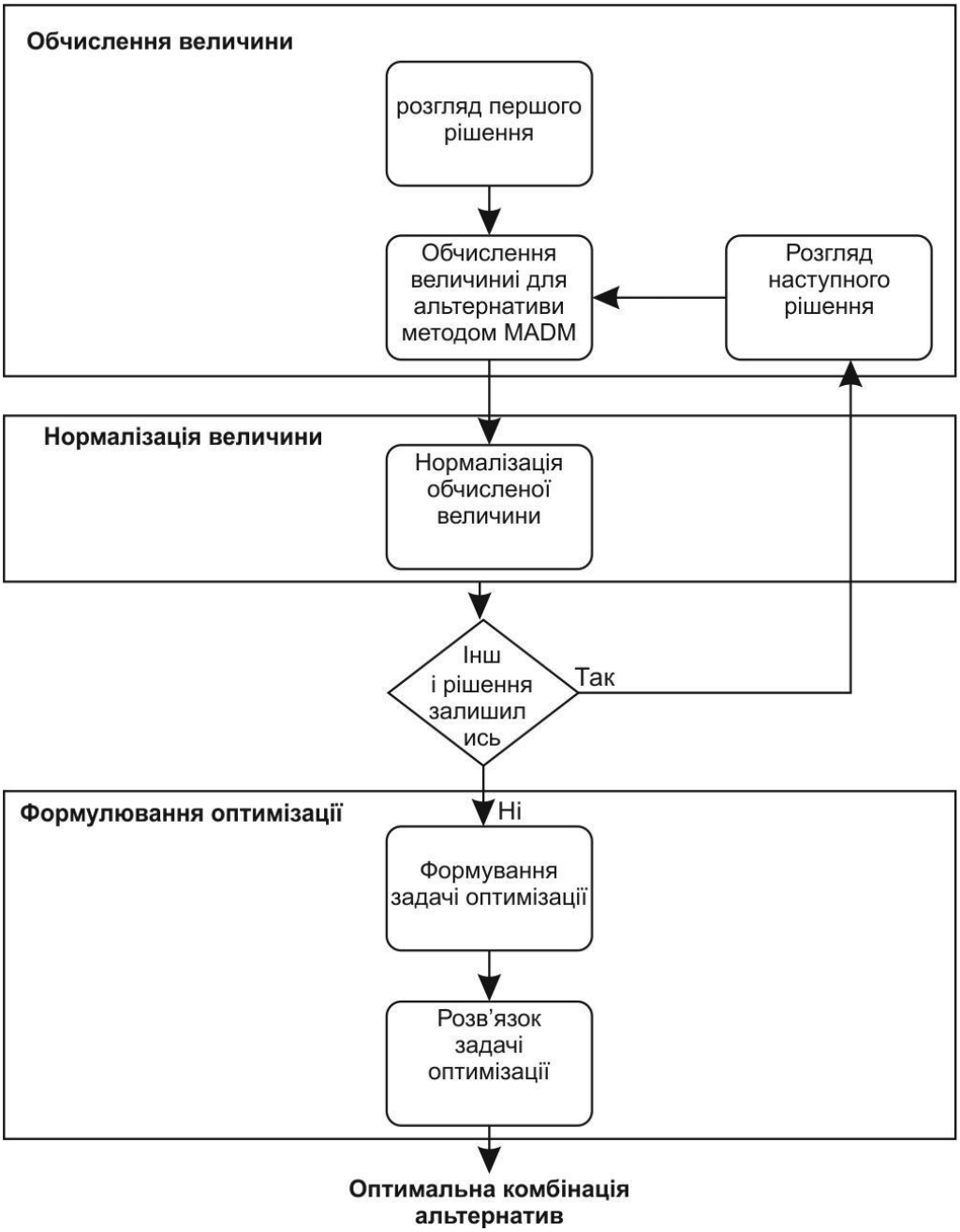

Рисунок 2.1 – Процес оцінювання ПА

Так повторюється для кожного рішення по розробці. Другий крок перетворює обчислені величини до нормалізованої форми з метою їх підготовки до третього кроку. Нарешті, третій крок формулює рівняння оптимізації так, щоби максимізувати величини, пов'язані з обраними альтренативами, з дотриманням сформульованих обмежень та взаємозалежностей.

2.3 Обчислення величин

Ми означаємо величину альтренативи розробки як ступінь, до котрої альтренатива задовольняє бажаним атрибутам якості. Для часткового рішення розробки потенційні альтренативи розробки оцінюються через множину атрибутів якості, пов'язаних з цим рішенням. Рисунок 2.2 описує процес обчислення величини для часткового рішення розробки. Вхідні дані для цього процесу подвійні:

1. Альтренативи розробки та їх відносна підтримка відповідних атрибутів якості. Одна альтренатива, для прикладу, може пропонувати високу надійність але низьку продуктивність. Інша може забезпечити високу продуктивність при втраті безпеки.

2. Переваги (пріоритети) пов'язаних атрибутів якості надаються різними замовниками. Наприклад, один замовним може бути більше зацікавлений у продуктивності та зчепленні за рахунок надійності та модифікованості. Інший замовник може бути більше зацікавлений лише у модифікованості та складності впровадження.

Для методики обчислення ми ґрунтуємося на методах Прийняття Рішення за Багатьма Атрибутами (ПРБА) – Multiple Attribute Decision Making (MADM) [17], котрий широко застосовується у різних сферах бізнесу. Ці методи дозволяють замовникам зробити рішення по преференціях стосовно наявних альтренатив, що характеризуються багатьма, часто конфліктуючими, атрибутами.

Зараз доступно декілька методів MADM, включаючи AHP, SAW, та ELECTRE. У цій роботі ми застосовуємо Метод Аналізу Ієрархій (MAI) [18]. Попри це запропонований метод є, однак, достатньо гнучким до акомодації інших методів на вибір різних замовників.

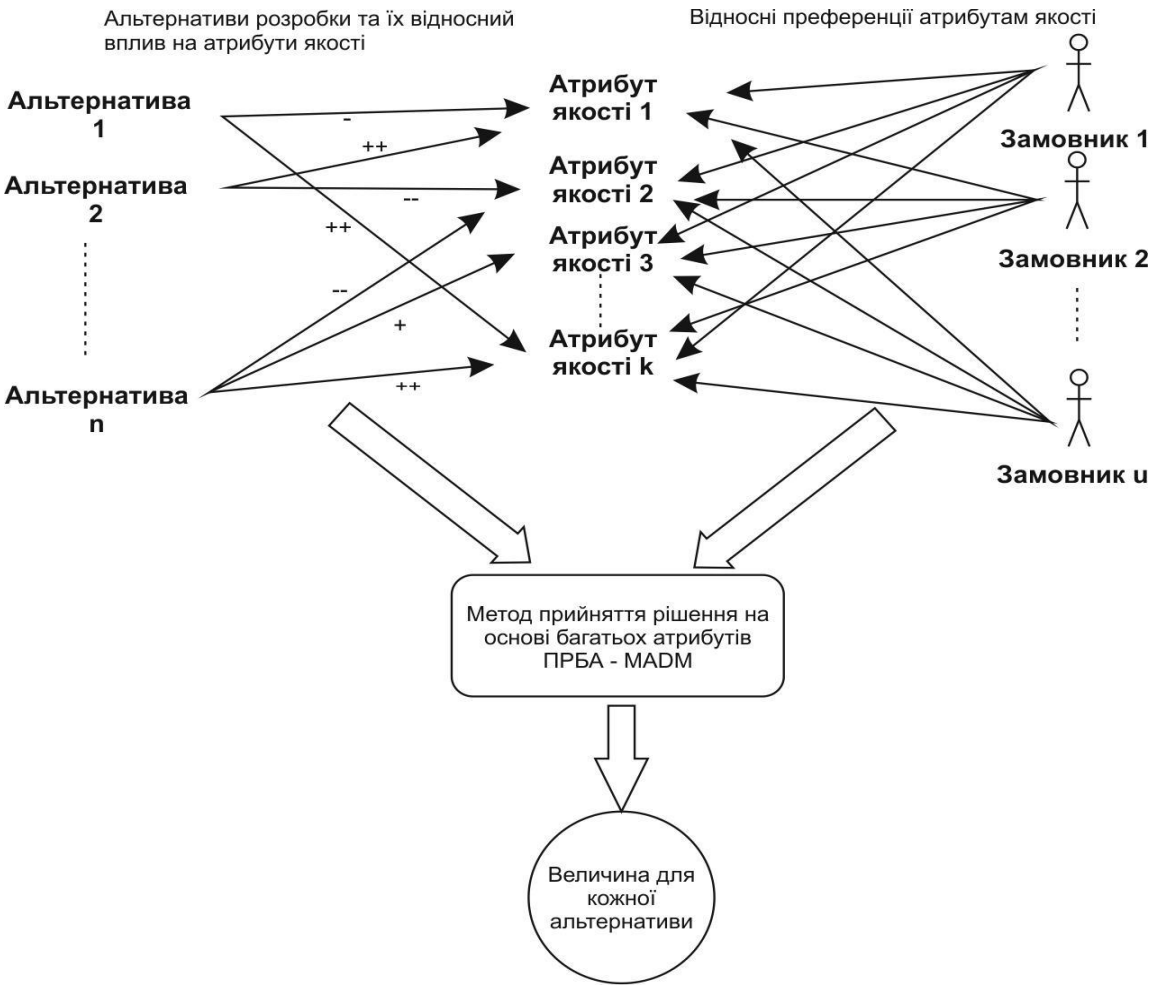

Рисунок 2.2 – Обчислення величини

MAI на відміну від інших методів MADM базується на відносних вагах (парних порівняннях) і таким чином менш чутливий до помилок оцінювання, спільних для інших методів, що використовують абсолютні величини.

MAI включає чотири основних кроки.

Підготовка: цей крок виділяє різні елементи, необхідні для процесу прийняття рішення для повноважного рішення. Це веде до ідентифікації альтренатив розробки, атрибутів якості, які використовуватимуться для оцінювання цих альтренатив, а також різних замовників, що братимуть участь у прийнятті рішення.

Зважування атрибутів якості: ціль цього кроку – визначити відносні ваги для кожного атрибуту якості. Для кожного рішення з розробки кожен замовник надає його преференції по відповідних атрибутах якості шляхом порівняння кожної пари цих атрибутів якості (Qa,Qb), застосовуючи шкалу ваг з таблиці 2.1.

Вона використовується для позначення того, наскільки важливий *Qa* у порівнянні з *Qb*. Зазначимо, що величини (2, 4, 6 та 8) показують компромісні (проміжні) значення між цими преференціями. Це означає, що для k атрибутів якості треба зробити $\frac{k(k-1)}{2}$ порівнянь кожним замовником. Переваги атрибутів якості для кожного замовника узагальнюються перед тим, як ми обчислюємо ваги атрибутів *Wz*, $1 \geq z \geq k$.

Зважування альтренатив для задоволення якості: далі ми визначаємо, як кожна альтренатива розробки підтримує відносні атрибути якості. Для кожного атрибуту якості *Qa* ми порівнюємо *n* потенційних альтренатив розробки попарно. Величини з таблиці 1 використовуються для присвоєння ваг для визначення того, як альтренатива *Ax* задловольняє атрибут *Qa* у порівнянні з альтренативою *Ay*. Потім ми можемо встановити відносну підтримку кожною альтренативою кожного атрибуту якості (вагу чи міру впливу кожної альтренативи на кожен атрибут якості). Це приводить нас до матриці ваг розміром *n* x *k*:

$$S = \left(S_{ax}; 1 \leq x \leq n, 1 \leq a \leq k\right), \qquad (2.1)$$

де кожен елемент (*x*,*a*) відповідає тому, як альтренатива *Ax* відносно підтримує атрибут якості *Qa*.

Таблиця 2.1 – Шкала ваг для MAI

Якщо A … , ніж B	**Числова вага**
Рівноважливі	1
Посередньо більш важливий	3
Сильно більше важливий	5
Дуже сильно важливіший	7
Беззаперечно важливіший	9

– Обчислення величини: тепер можна обчислити величину V_{ij} для і-ї альтренативи j-го рішення, використовуючи наступну формулу:

$$V_{ij} = \sum_{z=1}^{k} W_z S_{iz} \quad . \tag{2.2}$$

2.4 Нормалізація величин

Перед виконання третього кроку нам потрібно нормалізувати величини, отримані на попередньому кроці. Причиною цього є те, що числові характеристики альтренатив різних рішень потрібно додати на наступному кроці. Тому потрібно привести їх відносно однієї шкали. Для цього ми зважуємо різні рішення N_j відносно таким чином, щоби відобразити їхнє відносне значення для застосування. Природньо, що деякі рішення з розробки є більш важливими за інші і таким чином повинні отримати більшу вагу. Зробивши це, ми потім множимо отримані числові значення на вагу відповідного рішення:

$$V'_{ij} = N_j \cdot V_{ij} \, . \tag{2.3}$$

2.5 Формулювання оптимізації

На цьому кроці ми намагаємось максимізувати акумуляційне значення величини, яке представляє цільова функція. Це найкраще може бути сформульоване з застосовуванням амтематичного програмування.

Максимізуємо:

$$\sum_{j=1}^{m} \sum_{i=1}^{n_j} X_{ij} V'_{ij} \, . \tag{2.4}$$

За умови, якщо:

$$\forall j \in [1,...,m] : \sum_{i=1}^{n_j} X_{ij} = 1 \tag{2.5}$$

$$Cost\left(X_{i_1 1}, X_{i_2 2},..., X_{i_m m}\right) \le Constra\text{int}_{\text{cost}} \tag{2.6}$$

$$Time\left(X_{i_1 1}, X_{i_2 2},..., X_{i_m m}\right) \le Constra\,\text{int}_{time} \tag{2.7}$$

$$X_{ij} + X_{ab} \le 1, \quad j \ne b \tag{2.8}$$

Тут:

$m \geq 1$ означає число конструкторських рішень, що розглядаються.

$n_j \geq 2$ означає число альтренатив всередині рішення j.

$X_{ij} \in [0,1]$ де 1 означає, що альтренатива i вибрана для j-го рішення, а 0 означає, що ця альтренатива не вибрана.

V_{ij} означає нормалізовану величину для i-ї альтренативи j-го рішення.

$Constraint_{cost}$ означає обмеження по вартості.

$Constraint_{time}$ означає обмеження по часу.

$Cost\left(X_{i_1 1}, X_{i_2 2},..., X_{i_m m}\right)$ і $Time\left(X_{i_1 1}, X_{i_2 2},..., X_{i_m m}\right)$ є функціями, що приймають список альтренатив на вхід і обраховують очікувану вартість та час відповідно вимог до реалізації цієї комбінації.

Рівняння (2.4) максимізує акумулятивну величину вибором комбінації альтренатив, що дають найвищу можливу величину. Рівняння (2.5) гарантує, що лише одна альтренатива вибрана для кожного рішення. Рівняння (2.6) і (2.7) гарантують, що вибрані альтренативи не порушують обмеження часу та вартості. Відмітимо що функції часу та вартості є загальними, їх вигляд залежить від контексту проєкту, де вони застосовуються. Ці функції можуть, однак, використовуватись для загальних моделей оцінки, таких як COCOMO [17], для визначення потрібного часу та вартості. Нарешті рівняння 7 підсилює залежності між альтренативами з різних рішень, частково випадок, коли дві альтренативи не можуть бути обрані одночасно.

2.6 Короткий опис тестового проєкту

Проєкт The Glass Box (GB) [6] є частиною багаторічної дослідницької програми зі створення нових інструментів та технологій для інформаційних аналітиків. Сам GB є промисловою програмною системою, що впроваджена у робоче середовище інформаційних аналітиків. Його основна роль – це збір детальної інформації про діяльність робочих станцій користувачів під час збору інформації та виконання завдань аналізу. Ця інформація записується протягом довготривалого періоду та є доступною для ряду дослідницьких проєктів з лабораторій Північної Америки. Існує приблизно 15 окремих дослідницьких проєктів, що фінансуються спільною програмою.

Застосування GB пізніше планується впровадити як інтеграційну та тестову платформу для залучених дослідних проєктів. Від дослідницьких проєктів вимагається під'єднання їхнього програмного забезпечення до середовища GB та демонстрація їх можливостей у допомозі аналітикам при розв'язку реальних задач. Це вимагає негайної реєстрації дій аналітика, таких як відкриття документа чи виконання пошуку. Також для того, щоби зробити знання, згенеровані кожним дослідницьким інструментом, спільними, повинен бути наявний механізм для зберігання даних, згенерованих кожним інструментом та повідомлення інших інструментів про їх існування. Принцип роботи такого застосунку на рівні концепції зображено на рисунку 2.3.

Рисунок 2.3 відображає відношення між застосуванням GB та різними залученими замовниками. Інформаційні аналітики (Information Analysts) є первинними користувачами програми. Вона інтегрована у їх робоче середовище та забезпечує як прозорі, так і точні інструменти для запису дій. Команда розробників (GB Development Team) відповідальна за всі аспекти розробки GB, розвиток та впровадження. Команди дослідників (Research Teams) бачать GB як програмний засіб та репозиторій даних. Їм потрібно здійснювати доступ до даних в репозиторії та програмно інтегрувати їхні інструменти у середовище GB.

Фондова агенція (Funding Agency) відповідальна за затвердження планів розвитку та розподіл пов'язаних бюджетів, а також за загальний успіх програми.

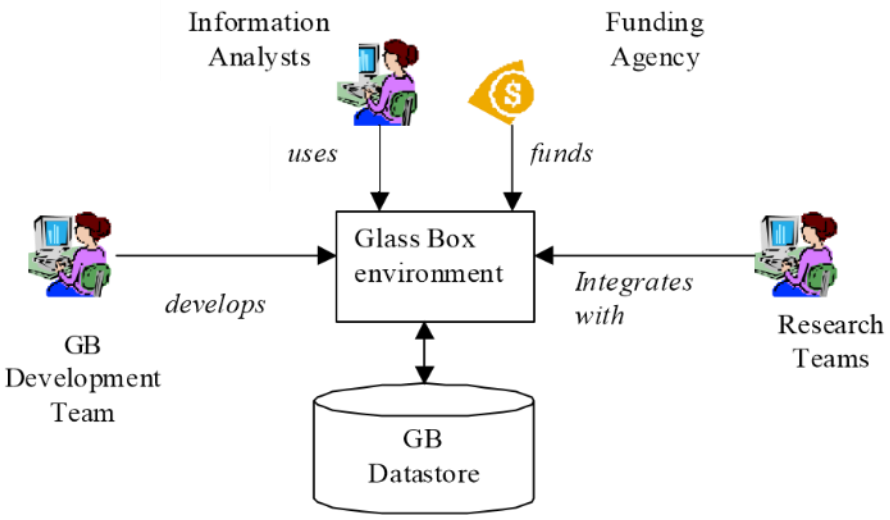

Рисунок 2.3 – Користувачі Glass box (Fund agency – фондова агенція)

Проєкт GB був розпочатий у 2002, а на початку 2003 р. перша версія програми була випущена та успішно впроваджена. Ця версія була в основному зосереджена на задоволенні потреб інформаційних аналітиків у зборі та зберіганні їх робочої діяльності.

Початкова версія GB представляла собою 2-х рівневу клієнт-серверну систему з застосовуванням бази даних, файлового сховища, документів, розпочатих та завершених завдань. Вона виконувалась окремо на кожній робочій станції користувача. Вночі скрипти вибирали дані з індивідуальних баз даних та переносили їх на центральне сховище даних для періодичного розподілення між командами дослідників.

Відразу після цього випуску розпочалось планування версії 2. В той час, як для версії 1 основним було створення середовища збору інформації для аналітиків, наступна версія мала значно ширший набір вимог. Коротко, це були такі вимоги:

– Забезпечення програмного доступу до бази даних GB для всіх команд дослідницького проєкту.

– Забезпечення механізму негайного інформування дослідницьких інструментів, коли аналітик виконує дію з набору його технологічних операцій.

– Забезпечити безпечний програмний доступ до середовища GB з віддаленого комп’ютера.

– Надавати можливість масштабування для підтримки великої кількості користувачів.

Додатковою вимогою було цей новий список функцій повинен бути втілений протягом одного року при встановленому бюджеті на розробку.

2.7 Аналіз прикладу застосування пропонованого підходу

В цьому розділі пропонується застосувати розроблений підхід до оцінювання ахрітектури (ПА) для аналізу більшості ахрітектурних рішень, зроблених під час розробки GB. Потім порівняємо результати, отримані з допомогою цього підходу, з рішеннями, які було в дійсності зроблено.

З цією метою було проведено декілька інтерв’ю з провідним архітектором.

Початково було щонайменше 9 ахрітектурних рішень, що піддались інтенсивному обговоренню та оцінюванню під час стадії розробки. Ці рішення показані на рисунку 2.4. Було обрано 5 рішень з 9-ти для включення до даного дослідження. Вони показані сірими прямокутниками.

Прямі стрілки на рисунку 2.4 показують взаємозалежності між різними рішеннями. Наприклад, Управління станами та рішення Ахрітектура є незалежними, в той час як Реєстрація подій та рішення Аутентифікація є залежними від рішення Ахрітектура. Ми використаємо терміни Головний та Підлеглий для опису цього типу взаємозв’язку. Наприклад, Рішення Гетерогенність є підлеглим для рішення Ахрітектура.

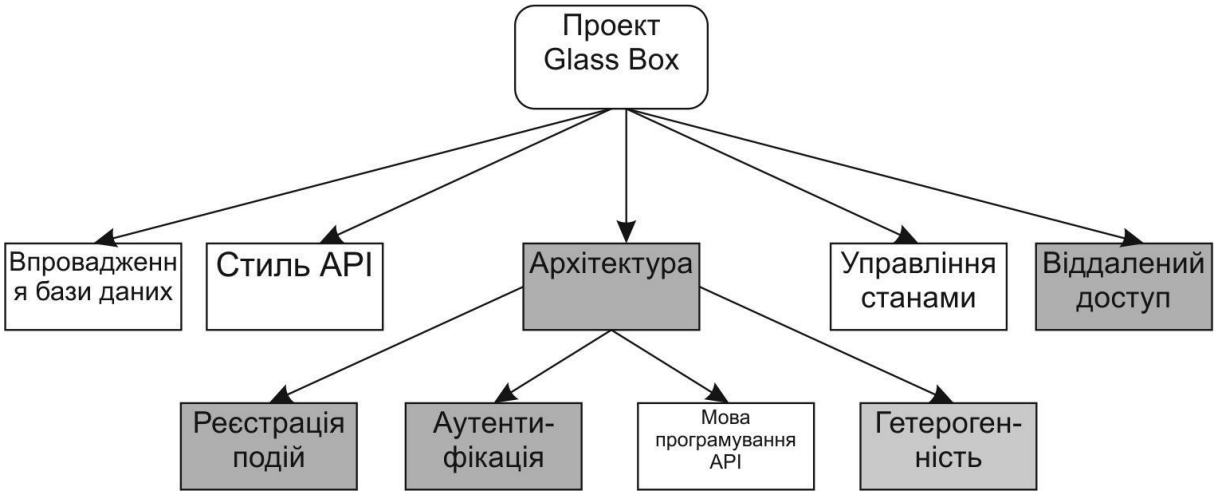

Рисунок 2.4 – Рішення з розробки GB та їх взаємозалежності

Список обраних рішень розробки показаний у таблиці 2.2 разом з відповідними альтренативами, які ми розглядали, відносними атрибутами якості та участю замовників у процесі прийняття рішення. Виділені альтренативи представляють реально здійснений вибір. Схематично ці ж ахрітектурні рішення показані у вигляді ієрархії на рисунка 2.5 та 2.6.

Для обробки взаємозалежностей, властивих проєкту, ми ідентифікували два типи взаємозалежності рішень:

– Залежність на базі альтренативи: тут альтренативи, доступні для підлеглого рішення, змінюються залежно від альтренатив, вибраних для його головного рішення. Приклади залежностей, показані на рисунку 5 та 6 відповідно між рішенням Ахрітектура та рішення Реєстрація подій, а також для рішень Ахрітектура та Аутентифікація. Далі, лише База даних (DB) є альтренативою для аутентифікації, якщо вибрано дворівневу ахрітектуру (TWOT).

– Залежність на базі контексту: Тут альтренативи, що підтримують певні атрибути якості (для підлеглого рішення), можуть змінюватись позитивно чи негативно залежно від альтренатив, вибраних їх головним рішенням. Це ми називаємо контекстом рішення. Наприклад, розглянемо альтренативу на базі мови Java гетерогенного рішення. Залежність тут походить від того факту, що атрибути якості "вартість" та "затрати на розробку" будуть відносно падати (тобто покращуватись), якщо вибрано трирівневу ахрітектуру J2EE (THTJ). Це

означає, що та сама альтренатива Java отримуватиме різні величини оцінки для різних трьох частин.

Таблиця 2.2 – Список вибраних рішень разом з їх альтренативами, атрибутами якості та замовниками.

Рішення розробки	**Альтренативи**	**Атрибути якості**	**Замовники**
Ахрітектура (ARCH)	• **3-рівнева з застосовуванням J2EE (THTJ)** • 3-рівнева з .Net (THTD) • 2-рівнева (TWOT) • COABS (COAB)	• модифікованість • масштабованість • продуктивність • вартість • Зусилля на розробку • переносимість • простота встановлення	• Команда розробників • Команди дослідників • Фінансова агенція
Реєстрація подій (EVNT)	• **Публікація-підписка з застосовуванням JMS** (JMS) • Публікація підписка • Тригери баз даних (TRGR) • COABS (COAB)	• Надійність • Продуктивність • Складність впровадження	• Команда розробників • Команди дослідників

Продовження таблиці 2.2

Рішення розробки	Альтренативи	Атрибути якості	Замовники
Аутентифікація (AUTH)	• **Безпека, базована на базі баз даних (DB)** • Безпека, базована на J2EE (J2EE) • Безпека, базована на .Net (.NET) • COABS (COAB)	• Складність впровадження • Простота розгортання та настройки	• Команда розробників • Команда дослідників
Віддалений доступ (RMAC)	• На базі бровзера (HTTP) • **Web служби (WEBS)** • Безпечні мережі (VPN)	• Продуктивність • Безпека • Модифікованість • Складність розгортання • Складність впровадження	• Команда розробників • Команди дослідників • Фінансова агенція
Підтримка API відмінних від Windows платформ (HETR)	• **мова Java (JAVA)** • бровзер (BROW) • мова C (C)	• Придатність до застосовування • Модифікованість • Вартість • Зусилля на розробку	• Команди дослідників • Команда розробників

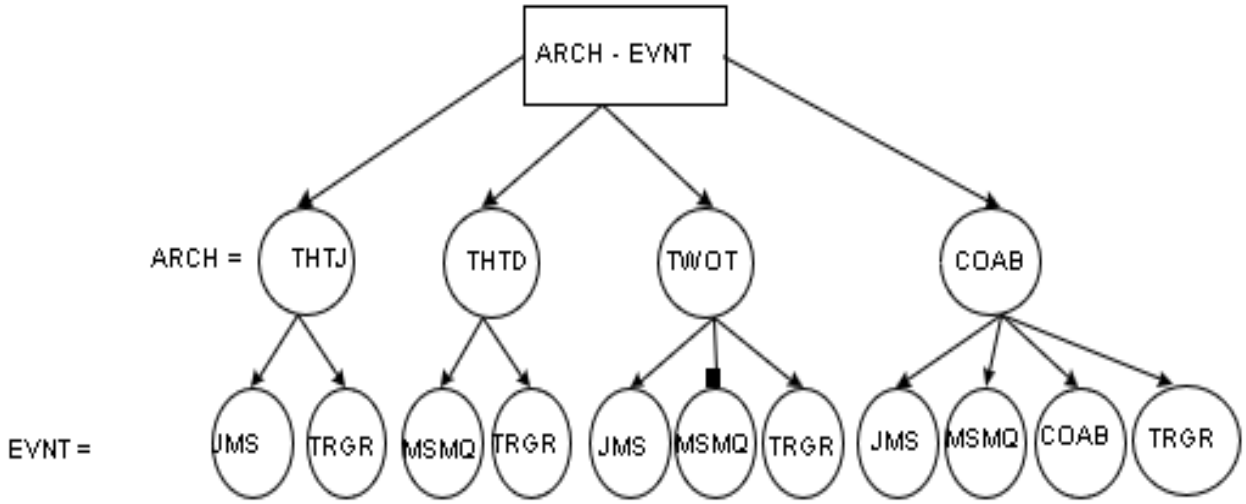

Рисунок 3.3 – Залежності для рішеннями ахрітектура на базі Реєстрації подій

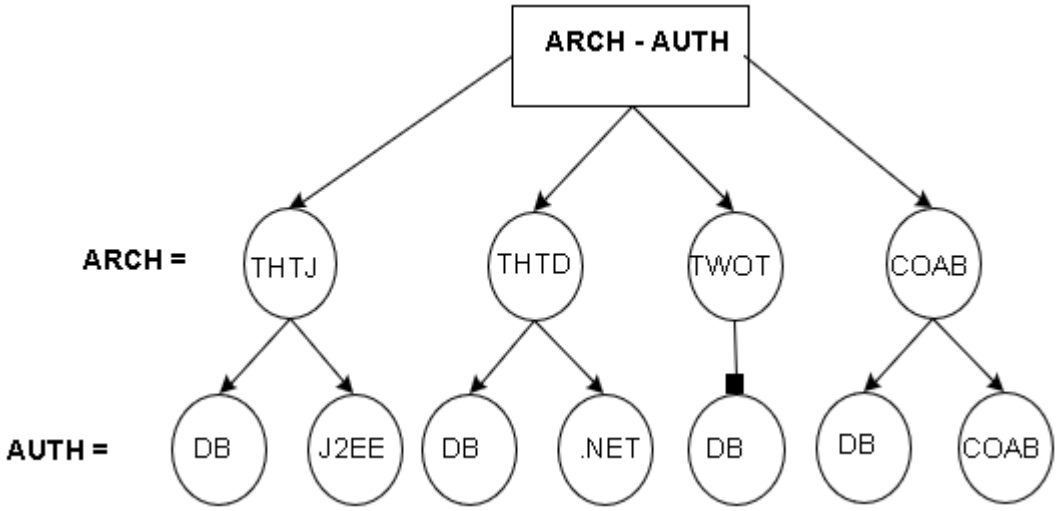

Рисунок 2.6 –Залежності для рішеннями ахрітектура на базі з Аутентифікацією

Після аналізу вибраних рішень та їх альтренатив і взаємозалежностей існувала 171 потенційна комбінація альтренатив для розгляду архітектором. На додачу, кожне рішення включало більше однієї групи замовників, кожна з яких надавала перевагу різним розв’язкам. Також проєкт був обмежений тривалістю один рік. Всі ці чинники зробили процес розробки надзвичайно складним.

2.8 Застосування підходу

2.8.1 Обчислення значення

На цьому кроці ми застосовуємо загальний інструмент MAI, ExpertChoice [6], для обчислення величин кожної альтренативи. Ми застосовуємо MAI до кожного рішення почергово. Для кожного рішення ми просимо головного архітектора надати нам преференції різних атрибутів якості, виражених на базі бачення різник замовників. Для економії місця покажемо лише, як було обраховано значення для рішення Ахрітектура (ARCH).

Таблиця 2.3 показує важливості (преференції) атрибутів якості для замовника – фондової агенції. У деяких випадках певним атрибутам якості не було надано важливості жодним замовником. Наприклад, команди замовників не брали до уваги вартість розробки, що було обов'язковим для команди розробників. Ми обробили такий випадок через присвоєння їм найнижчої можливої ваги (1 чи 9) у порівнянні з будь-яким іншим атрибутом якості.

Таблиця 2.3 – Преференції атрибутів якості, надані замовником - фондовою агенцією для рішення ARCH

Замовник	**1-й атрибут якості**	**2-й Атрибут якості**	**Кількісна вага**
Фондова агенція	Модифікованість	Масштабованість	5
	Модифікованість	Продуктивність	2
	Модифікованість	Вартість	1
	Модифікованість	Затрати на розробку	1
	Модифікованість	Переносимість	3
	Модифікованість	Легкість встановл.	1
	Масштабованість	Продуктивність	1/4
	Масштабованість	Вартість	1/5
	Масштабованість	Затрати на розробку	1/5

Продовження таблиці 2.3

Замовник	**1-й атрибут якості**	**2-й Атрибут якості**	**Кількісна вага**
	Масштабованість	Переносимість	1/2
	Масштабованість	Легкість встановл.	1/2
	Продуктивність	Вартість	1/5
	Продуктивність	Затрати на розробку	1/5
	Продуктивність	Переносимість	2
	Продуктивність	Легкість становл.	1
	Вартість	Затрати на розробку	1
	Вартість	Переносимість	5
	Вартість	Легкість встановл.	4
	Затрати на розробку	Переносимість	5
	Затрати на розробку	Легкість встановл.	4
	Переносимість	Легкість встановл.	1/2

Після збору преференція якості від замовників ми побачили, що вони змінюються від замовника до замовника, що відбивається у різних значеннях результуючих пріоритетів атрибутів якості.

Це показано у таблиці 2.4. Різні преференції від замовників були потім агреговані шляхом обчислення середнього геометричного значення для кожного атрибуту якості, і ця вага атрибутів показана в останньому стовпці таблиці 2.4.

Таблиця 2.4 – Ваги атрибутів якості, обраховні для кожного замовника для рішення ARCH

Атрибути якості	**Замовники**			**Агреговане значення**
	Розробники	**Дослідники**	**Фондова агенція**	
Модифікованість	0,216	0,294	0,184	0,280
Масштабованість	0,087	0,092	0,038	0,082
Продуктивність	0,052	0,117	0,087	0,097
Вартість	0,245	0,019	0,272	0,135
Затрати на розробку	0,245	0,019	0,272	0,135
Переносимість	0,050	0,155	0,053	0,094
Легкість встановлення	0,106	0,304	0,093	0,177

Потім ми попросили головного архітектора надати нам рівнями задоволення (підтримки) кожною альтренативою всіх атрибуті якості.

Ці дані були отримані лише з токи зору розробників, оскільки вони вимагають технічних знань, якими володіють лише розробники. Таблиця 2.5 показує, парне порівняння альтренатив по відношенню до атрибуту якості Переносимість.

Цікаво, що деякі парні порівняння було досить важко зважити – наприклад, тих, що пов’язані з COABS [4] по відношенню до певних атрибутів якості, таких як продуктивність та затрати на розробку. Головний архітектор був не в змозі визначити підходящу вагу через те, що розробники мали обмежений досвід з застосовування технології COABS та не було в наявності надійних критеріїв для порівняння цього власного рішення з іншими альтренативами. Ми обробили такі порівняння шляхом присвоєння їм ваги 1 з припущенням, що вони були однаково сильними стосовно досліджуваних атрибутів якості.

Таблиця 2.5 – Парні порівняння для альтренатив ARCH по відношенню до атрибуту Портативність

Атрибут якості	**1-[a] Альтренатива**	**2-[a] Альтренатива**	**Кількісна вага**
Портативність	THTJ	THTD	9
	THTJ	TWOT	9
	THTJ	COAB	1
	THTD	TWOT	1
	THTD	COAB	1/9
	TWOT	COAB	1/9

Використовуючи MAI, ці дані були перетворені у відносні ваги, показані у таблиці 2.6. Використовуючи цю таблицю, ми можемо сказати, як кожна з альтренатив відносно задовільняє (підтримує) кожен атрибут якості. Наприклад, трирівнева J2EE виглядає найкращою альтренативою стосовно задоволення модифікованості.

Застосовуючи формулу (2.2) до результатів таблиці 6 та агрегованих ваг з таблиці 2.4, ми отримали наступні величини для рішення ARCH, показані у таблиці 2.7. Аналогічно ми обрахували величини для всіх решту рішень, показаних у таблиці 2.7. Зазначимо, що альтренатива JAVA для рішення Гетерогенність продубльовано для вирішення базованої на контексті залежності з THTJ.

2.9 Нормалізація показників якості

На цьому кроці ми попросили головного архітектора зважити різні рішення відносно таким чином, щоби визначити їхню відносну важливість для застосування GB. Рисунок 2.7 показує шкалу ваг для різних рішень за 10-ти бальною шкалою.

Таблиця 2.7 – Ваги альтренатив ARCH

Атрибути якості	**Альтренативи**			
	THTJ	**THTD**	**TWOT**	**COAB**
Модифікованість	0,521	0,182	0,046	0,250
Масштабованість	0,402	0,402	0,054	0,143
Продуктивність	0,204	0,204	0,347	0,246
Вартість	0,166	0,120	0,487	0,227
Затрати на розробку	0,152	0,110	0,515	0,223
Портативність	0,450	0,050	0,050	0,450
Легкість встановлення	0,168	0,368	0,256	0,208

Потім ми перемножили величину оцінки, отриману на попередньому кроці на відповідну вагу, використовуючи формулу (2.3), в результаті нормалізуючи величини, наведені в останньому стовпці таблиці 2.7.

2.10 Формулювання оптимізації

Перед формулюванням задачі оптимізації нам потрібно означити обмеження для GB, а також те, як кожна альтренатива впливає на ці обмеження.

GB версії 2 була обмежена однорічним графіком розробки. На базі обговорення з головним архітектором здавалось, що досить важко (майже нереально) точно означити час, необхідний для впровадження кожної альтренативи окремо. Однак, виявляється, що у деяких випадках було можливо визначити, чи порушить кожна альтренатива часові обмеження, без точного зазначення, як довго вона впроваджуватиметься.

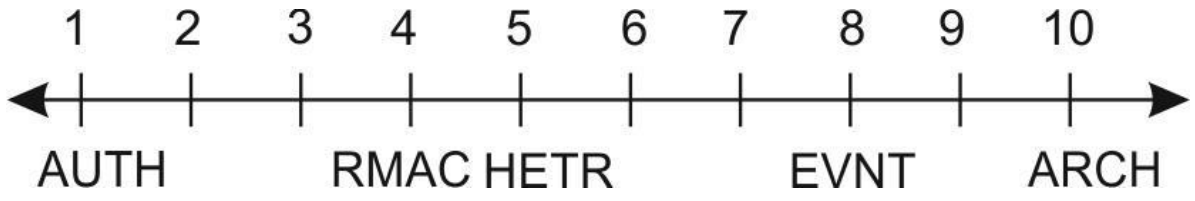

Рисунок 2.7 – Ваги для різних рішень

Так було у випадку з альтренативою віддаленого доступу через VPN та з альтренативою аутентифікації на базі J2EE, для яких була впевненість, що їх впровадження кожна з них перевищить обмеження розкладу.

На відміну від цих двох альтренатив, всі інші альтренативи були в межах часових обмежень. Для врахування цієї ситуації ми використали наступну формулу:

$$Time\left(X_{i_1 1}, X_{i_2 2}, ..., X_{i_m m}\right) = \begin{cases} > 12 \text{ } місяців, \text{ } якщо \text{ } X_{34} = 1 \text{ } або \text{ } X_{23} = 1 \\ \leq 12 \text{ } місяців \text{ } в \text{ } решті \text{ } випадків \end{cases} \quad (2.7)$$

Функція *Time*() повертатиме число, більше за 12 місяців, якщо комбінація складається з X_{34} (VPN Remote Access) або X_{23} (J2EE-Based Authentication). X_{34} читається, як вибір третьої альтренативи четвертого рішення. Це може призвести до порушення обмеження згідно формули (2.7).

Таблиця 2.7 – Отримані величини для альтренатив

Рішення	**Альтренатива**	**Величина**	**Нормалізована величина**
Ахрітектура (ARCH)	THTJ	0,314	3,14
	THTD	0,205	2,05
	TWOT	0,236	2,36
	COAB	0,246	2,46
Реєстрація подій (EVNT)	JMS	0,258	2,064
	MSMQ	0,272	2,176
	TRGR	0,229	1,832
	COAB	0,241	1,928
Аутентифікація (AUTH)	DB	0,215	0,215
	J2EE	0,358	0,358
	.NET	0,223	0,223

Продовження таблиці 2.7

Рішення	Альтренатива	Величина	Нормалізована величина
	COAB	0,204	0,204
Віддалений доступ (RMAC)	HTTP	0,326	1,304
	WEBS	0,227	0,908
	VPN	0,446	1,784
Підтримка API відмінних від windows платформ (HETR)	JAVA	0,257	1,285
	BROW	0,294	1,47
	C	0,155	0,775
	JAVA з THTJ	0,295	1,475

Застосовуючи рівняння оптимізації, ми отримали наступну комбінацію: Трирівнева для Ахрітектури, JMS для Реєстрації подій, Бази даних для аутентифікації, Мова Java для Гетерогенності та На базі бровзера для віддаленого доступу. Цей список був рекомендований як оптимальний для ахрітектури з найвищим акумульованим значенням 8,198. Ця комбінація змінюється лише для випадку рішення Віддаленого доступу з альтренатив, що були вибрані у проєкті GB (див. таблицю 2.3).

2.11 Інтерпретація результатів застосування пропонованого методу

Далі викладемо основні висновки даного дослідження:

– Рекомендована як кандидат ахрітектура показує прийнятний рівень точності для пропонованого підходу, оскільки лише одне рішення (RMAC) було відмінне від реальної розробки.

– Огляд базованих на контексті залежностей може привести до рішень, що ідуть відразу після оптимальних. Це був випадок для рішення Гетерогенність, де альтренатива Бровзер була би першою за значенням, якби не розглядались контекстні впливи від головного рішення (ахрітектура).

– Базуючись на зв’язку з головним архітектором, агрегація преференцій атрибутів якості усіх замовників (для кожного рішення) видається як обґрунтований компроміс між конфліктуючими цілями якості замовників.

– В процесі вибору мають бути взяті до уваги обмеження проєкту. Часові обмеження виключили дві альтренативи з двох різних рішень (Аутентифікація та Віддалений доступ), хоча вони були поставлені першими у їх відповідних рішеннях.

– Зворотний зв’язок до підходу від головного архітектора GB був позитивний. Систематичний та добре відображений процес прийняття рішення скоріше за все був помічний для розробників GB в частині розробки ахрітектури. Це могло б спричинити багато дискусій при виявленні альтренатив, та могло б допомогти комунікації всередині колективу та оцінюванні рішень замовниками.

2.12 Покращення розглянутого методу

Головним внеском пропонованого методу є контрольована методологічна підтримка, яку він пропонує для процесу розробки ахрітектури. Метод не призначений для заміни архітектора при прийнятті рішень. Скоріше він допомагає архітектору у:

– Прийнятті інформованого ахрітектурного рішення: через кількісну оцінку якісних преференцій замовників та підтримки атрибутів якості альтренативами розробки метод надає формальний шлях до раціоналізації та оцінювання рішень. Він також пропонує строгий але простий шлях обгрунтування якості ахрітектури на ранніх стадіях розробки шляхом явного оцінювання атрибутів якості.

– Побудова розподіленої ахрітектури: пропонований метод надає контрольований інженерний підхід до розробки розподілених програмних ахрітектур. На детальному рівні це дозволяє архітекторам вивчати і оцінювати усі ахрітектури-претенденти та досліджувати компроміси між наявними альтренативами розробки.

– Спрощення складності розробки. Пропонований метод допомагає зробити процес розробки ахрітектури більш систематичним через формальну обробку різних рішень розробки, альтренатив, взаємозалежностей, цілей якості замовників та обмежень проєкту. Через управління всіх чинників, що впливають на процес розробки, він також допомагає визначити оптимальну комбінацію альтренатив розробки.

– Обробка часток різних замовників: Через агрегацію преференцій замовників по атрибутах якості пропонований метод досягає балансу між конфліктуючими цілями якості у замовників. Додатково метод полегшує комунікацію ахрітектурних рішень між різними замовниками, підвищуючи таким чином точність зробленого рішення.

2.13 Обмеження методу

Під час дослідження було виявлено деякі обмеження:

– MAI має обмеження у 9-точкову шкалу зважування, що веде до неузгодженості. ЦЕ спостерігалось під час порівняння альтренатив web-сервісу та VPN відносно атрибуту якості Складність розгортання. Ми присвоїли тут вагу 9 як найбільшу можливу, але якби було можна, то ця вага мала б бути значно вищою.

– Невизначеність у оцінках не була формально опрацьована, зокрема оцінювання з застосовуванням альтренативи COABS. Можна стверджувати, що обробка невизначеностей у оцінках могла би підсилити метод у видачі більш точних результатів.

– Посилаючись на той спосіб, яким було виконано прийняття рішення про Віддалений доступ, виявляється, що було використано когнітивний підхід. Таким чином, різні підходи до прийняття рішень, такі як профілі можуть бути прийнятними для певних рішень розробки.

– Як написано у вище, було нереально очікувати точних оцінок часу, необхідного для впровадження кожної альтренативи. Однак, було можливим

сказати на базі досвіду, чи окрема альтренатива потребує більше/менше, ніж 12 місяців для впровадження. Можливе рішення – це мати відносні ваги відносно часових обмежень, котрі називаються рейтингом ідеальних величин [16].

– Рівень цілісності оцінювання не був виміряний перед обчисленням величин для альтренатив. Вимірювання рівня цілісності оцінок замовників допоможе забезпечити точність оцінювання. У випадку нецілісних оцінок замовники можуть потребувати додаткових зважувань.

2.14 Можливі покращення методу

У цій роботі пропонується метод, як системний підхід для полегшення розробки ахрітектури розподілених програмних застосувань. Використовуючи техніку оптимізації він намагається визначити оптимальну комбінацію альтренатив розробки, яка найкраще задовільняє цілі якості замовників та обмежень проєкту. Для забезпечення перевірки підходу ми застосували його для післяпроєктного (після завершення проєкту) аналізу вже впровадженого рішення, і результати показали високий рівень точності пропонованого підходу.

Як сказано у минулому розділі було виявлено деякі обмеження в отриманих результатах. Ми плануємо подолати ці обмеження. Крім того, оскільки наше дослідження було проведено як післяпроєктне (по завершенні проєкту), то існує можливість спотворення (зсуву) тверджень головного архітектора на користь прийнятих в проєкті рішень, що можливо вплинуло на оцінювання в MAI. Ми також спростили задачу проєктування шляхом вибору лише підмножини розробок в рамках проєкту GB. Таким чином, ми маємо намір провести інший експеримент під час розробки ахрітектури системи. Ми також плануємо розробити інструмент на базі пропонованого методу для підтримки процесу розробки ахрітектури.

3 ПРАКТИЧНА РЕАЛІЗАЦІЯ

Сучасний рівень розвитку технологій проєктування програмних систем (ПС), дозволяє за короткий час створювати складні багатофункціональні програмні комплекси.

Головною метою цих технологій є забезпечення функціональних вимог до ПС, а виконання вимог якості контролюється на останній стадії життєвого циклу (ЖЦ), до готового програмного продукту. При невиконанні вимог якості, вносяться зміни в проміжні продукти на всіх попередніх стадіях ЖЦ, що призводить до перевитрат бюджету проєкту, подовження термінів його виконання, а можливо і до відхилення проєкту замовником [1].

В стандарті з процесів життєвого циклу ПС ISO 12207 передбачено два процеси, які мають відношення до якості, це процес "забезпечення гарантії якості" та процес "управління якістю". Цілями першого процесу є забезпечення гарантії якості ПС шляхом впровадження в технології розробки ПС стандартів якості та відповідних процедур. Процес "управління якістю" призначений для реалізації моніторингу забезпечення вимог замовника ПС до якості.

Процес "забезпечення гарантії якості" інтегрується з процесом "управління якістю", таким чином, що результати моніторингу якості використовуються для планування дій по задоволенню вимог якості. Причому, ці процеси повинні реалізуватися на всіх стадіях ЖЦ на базі користувацьких вимог, тобто необхідно виконати процедури комунікації вимог [2]. При цьому виникає ряд проблем, повязаних із складом та формою представлення вимог до проміжних продуктів, та визначення зв'язків між вимогами сусідніх стадій ЖЦ.

Для розробки вимог якості до ахрітектури можна використовувати моделі вимог якості у використанні (ЯВ), зовнішньої та внутрішньої якості (ЗЯ, ВЯ) стандарту ISO 25010 [3]. Це дає змогу сформулювати вимоги до ПС в цілому, вимоги до ахрітектури та до її компонентів в єдиній уніфікованій формі. Для

визначення зв'язків між вимогами можна використати методи статистики, або експертні методи [5].

В даній статті містяться результати досліджень по проєктуванню ахрітектури web-застосувань на базі зазначеного підходу.

3.1 Сучасний стан технології проєктування ахрітектури ПС

3.1.1 Проєктування ахрітектури на базі патернів

Наступним етапом після розробки вимог є проєктування ахрітектури ПС. Цей етап є дуже важливим, оскільки на ньому вирішується задача задоволення як функціональних вимог, так і вимог якості.

Найбільш широко, в даний час, при проєктуванні ахрітектури використовується концепція типових рішень, яка базується на виборі з репозиторію ахрітектур ПС, отриманих з практики, найбільш підходящої для реалізації поставлених функціональних вимог до ПС.

М. Шоу і Д. Гарлан в [5] класифікували ахрітектури ПС, виділивши п’ять груп по реалізації ними певного набору стандартних загальних функцій. Е. Гамма в [6] розробив ахрітектурні зразки проєктування (патерни), які представляють комбінацію компонентів, зазвичай класів або об’єктів, що вирішують певні проєктувальні задачі. Зразки проєктування в [5] розділені на структурну, креаційну і поведінкову категорії. Структурні зразки моделюють способи представлення об’єктів, такі як дерева або звязні списки. Вони дозволяють користуватися множиною об’єктів як одним цілим, тому надають певні зручності.

Креаційні зразки пов’язані із методом створення складних об’єктів, таких як, лабіринти і дерева.

Поведінкові зразки проєктування дозволяють відстежувати поведінку об’єктів, наприклад видавати звіт про колекцію об’єктів в певному порядку.

В загальному випадку патерн проєктування представляє собою взаємодію об’єктів і класів, пристосованих для розв’язання загальної задачі проєктування,

яка вирішується в конкретних умовах. Слід зазначити ,що в інших відомих технологіях проєктування ахрітектури ПС, таких як, MVC (Model/View/Controller) [7] або модель шарів (layers) [8] теж достатньо плідно використовується концепція патернів.

3.1.2 Вибір ахрітектури ПС з врахуванням показників якості

Оскільки при використанні концепції патернів вибирається декілька альтренативних ахрітектур, які задовольняють функціональним вимогам, то вибір конкретної ахрітектури проводиться шляхом оптимізації, з врахуванням вимог якості.

Одним з найбільш відомих підходів до вибору ахрітектурних рішень з врахуванням вимог якості є технологія ADD - ATAM (Attribute - Driven Design, Architecture Trade of Analis Method) [9]. Перша частина технології ADD призначена для вибору ахрітектурних зразків проєктування шляхом розробки тактик, кожна з яких призначена для реалізації певного атрибута якості, причому в кожному зразку реалізується декілька тактик. Тобто тактики представляють собою вид проєктного рішення, яке впливає на керування реакцією за даним атрибутом якості. ADD - є рекурсивним процесом декомпозиції, на кожному з етапів якого відбувається відбір тактик і ахрітектурних зразків, які задовольняють певним сценаріям якості.

Реалізація відібраних тактик і ахрітектурних зразків, для вибору найкращої ахрітектури виконується методом ATAM. Метод ATAM реалізується за чотири етапи, перші два з яких представляють оцінні операції і є аналітичними. На цих етапах виконується виявлення та аналіз ахрітектурних методик і патернів, а також їх відбір на базі так званого "дерева корисності".

Ця технологія має ряд недоліків методологічного, організаційного та технічного характеру.

По-перше, в ній використовуються нестандартизовані та неуніфіковані показники якості, і метрики, що може породжувати неоднозначність їх трактувань. Також в технології відсутня процедура комунікації вимог якості,

сформульованих до ПС, на вимоги до ахрітектури, що може приводити до некоректних результатів.

Являючись досить об'ємною і вартісною технологією ADD - ATAM може використовуватись лише великими колективами розробників для розробки великих програмних проєктів.

3.2 Застосовування моделей якості для розробки вимог

Питання розробки вимог якості до ПС на базі моделей якості, розглядались в ряді публікацій [12], [13]. Тут розглядаються моделі трьох категорій якості, а саме ЯВ, зовнішньої та ВЯ. На базі моделі ЯВ розробляються загальні вимоги якості користувача, або замовника. Модель вимог ЯВ ПС можна представити у вигляді [4]

$$V_{use} = \{H^u_i, A^u_{ik}, C^u_{ik}, M^u_{ik}\}, \; i \in N_u, \; K = 1, S_i , \qquad (3.1)$$

тут H_{ui} - характеристика ЯВ;

A_{uik} - атрибут характеристики якості;

C_{uik} - обмеження на значення атрибута;

M_{uik} - метрика атрибута.

Характеристики і метрики підбираються із стандарту [13], а атрибути і обмеження із вимог користувача та аналізу предметної області.

Вимоги ЗЯ представляються у вигляді структури моделі ЗЯ і інтерпретуються як вимоги до ПС в цілому, в тому числі і до ахрітектури. Ці вимоги записуються у вигляді

$$V_{ex} = \{H^e_i, П^e_{ik}, A^e_{ik}, C^e_{ik}, M^e_{ik}\}, \; i \in N_e, \; K = 1, R_i \qquad (3.2)$$

тут Hei - характеристики;

Peik - підхарактеристики ЗЯ;

Aeik, Ceik, Meik - атрибути, обмеження та метрики відповідно.

Комунікація (трасування) вимог якості (3.1) на структуру вимог (3.2) виконується з застосовуванням методу SQFD [4]. На базі отриманих вимог ЗЯ формулюються вимоги якості до ахрітектури.

3.3 Розробка вимог якості до web – застосувань

Питання оцінювання якості web – застосувань досліджувались в ряді публікацій [10], [14]. Однак в них досліджувались лише певні аспекти якості, а не весь спектр характеристик, причому були використані неуніфіковані і нестандартизовані характеристики якості, що ускладнює порівняння отриманих результатів, та застосовування їх на практиці. В роботі [10] була проведена систематизація характеристик якості web – застосувань шляхом віднесення їх до наступних груп:

- технічні критерії надійності і безпеки;
- технічні критерії швидкодії і оптимізації трафіка;
- системні критерії;
- психологічні критерії;
- естетичні та художні.

Проєктування якісних web – застосувань на базі технічних критеріїв не викликає труднощів і для кожного класу web – систем визначено типові проєктні рішення. але врахування додаткових критеріїв викликає значні ускладнення, повязані з їх конфліктністю, повною схожістю критеріїв з різних груп та інше.

Хоча web – системи відрізняються великим різноманіттям, для них можна розробити систему критеріїв якості, загальну для всіх систем. Одним з шляхів вирішення цієї задачі є застосовування моделі якості стандарту ISO 25010, яка містить повний набір уніфікованих та стандартизованих характеристик і метрик якості. Наявність в стандарті трьох типів моделей якості дає можливість віднести їх до відповідних стадій життєвого циклу web і реалізувати процедури комунікації вимог.

Перейдемо до побудови моделі якості, яка відображає вимоги ЯВ для web-застосувань. Виходячи із загальних потреб користувачів і замовників, вимоги якості до web-ресурсів можна сформулювати наступною сукупністю критеріїв:

- Доступність.
- Можливість застосовування системи навігації.
- Можливість застосовування системи пошуку інформації.
- Зрозумілість структури сайту.
- Швидкість надання інформації.
- Точність надання інформації.
- Естетичне оформлення сайту.
- Скорочення часових ресурсів на виконання поставлених завдань.
- Скорочення фінансових затрат на виконання поставлених задач.
- Можливість застосовування системи ведення відвідуваності сайту.
- Можливість застосовування системи для ведення статистики отриманих послуг.
- Відповідність сайту галузевим чи міжнародним стандартам.
- Безперебійна робота протягом визначеного періоду часу.
- Безпечність зберігання та контролю над даними.
- Безпека користувачів.
- Надійність сайту (захист від атак ззовні).
- Наявність засобів наповнення інформаційного контенту і т.д.

Визначивши потреби та обмеження відобразимо атрибути на характеристики і метрики моделі ЯВ $\{H_i^u, M_{ij}^u\}\, i = \overline{1,4},\ j = \overline{1,B_i}$

Для відображення атрибутів на характеристики моделі і вибору відповідних метрик проведемо їх класифікацію. У результаті отримаємо стандартизоване представлення вимог користувачів бізнес-системи у вигляді моделі ЯВ. Для зручності застосовування в технології проєктування вимоги можна представити у вигляді шаблону. Наведемо приклад опису для атрибута "Можливість застосовування системи навігації". Провівши класифікацію цього атрибута, заповнення полів шаблону матиме вигляд як показано у таблиці 3.1.

Таблиця 3.1 – Опис атрибуту якості

Елемент шаблону	**Екземпляр елементу**
Характеристика	**Задоволеність**
Підхарактеристика	**-**
Назва атрибуту	**Можливість застосовування системи навігації**
Визначення атрибуту	Даний атрибут визначає необхідність наявності системи навігації для руху по сторінках web-ресурсу
Мета/Мотивація	Призначення атрибута полягає у реалізації можливості перегляду сторінок сайту користувачем з координацією переміщень по сайту
Шкала вимірювань	Тип шкали порядковий (присутність, відсутність)
Процедура визначення, протокол, Х	Процедура визначення атрибута – аналіз предметної області, потреба користувача.
	Примітки
Тип збору даних та підрахунку	Тип збору атрибута ручний
Метрика	Абсолютна продуктивність
Пріоритетність	0,7

Нижче наведемо специфікацію вимог ЯВ до web – сайтів структуровані відповідно моделі якості:

1. Ефективність

– Доступність.

– Точність надання інформації.

– Зрозумілість структури сайту.

2. Продуктивність

– Швидкість надання інформації

– Економія часових ресурсів.

– Економія фінансових затрат.

3. Безпечність

– Безпечність зберігання та контролю над даними.

– Безперебійна робота протягом визначеного періоду часу.

– Безпека користувачів.

– Надійність сайту.

4. Задоволеність

– Можливість застосовування системи навігації.

– Зрозумілість структури сайту.

– Естетичне оформлення сайту.

– Можливість застосовування системи ведення відвідуваності сайту.

– Можливість застосовування системи для ведення статистики отриманих послуг.

– Відповідність сайту галузевим чи міжнародним стандартам.

– Наявність засобів наповнення інформаційного контенту

Отже, у наведеній специфікації вимог до web-сайтів, які сформовані на базі моделі якості, в уніфікованій формі подано потреби замовника та користувачів ресурсів web-простору. Застосування запропонованої моделі дозволяє систематизувати вимоги до програмного забезпечення, на відміну від технологій, які наведені у [10]. В цій технології відсутня чітка структуризація вимог, тобто фактично неможливо встановити чітку приналежність вимоги до потреб замовника і до певного фізичного модуля ахрітектури, де вона буде реалізована. Крім того, такий підхід до розробки вимог не є уніфікованим і в ньому суттєвий вплив має субєктивний фактор.

3.4 Побудова моделі ЗЯ та виділення вимог до ахрітектури

Для розробки специфікації вимог до ахрітектури у термінах моделі ЗЯ web-сайтів використано каскадну процедуру проєктування [12]. Для цього проведемо аналіз характеристик $H_i^u,\ \ i \in N_u^k,\ K = \overline{1, S_i}$, атрибутів $A_{ik}^u,\ \ i \in N_u^k,\ K = \overline{1, S_i}$ та метрик $M_{ik}^u,\ \ i \in N_u^k,\ K = \overline{1, S_i}$ моделі ЯВ.

Для прикладу наведемо відображення атрибута "Можливість застосовування системи навігації" на атрибути моделі ЗЯ для web-застосувань. Аналізуючи системи навігації web-сайтів, їх можна розділити на два типи:

головна навігація та контекстна навігація. Виходячи з цього, атрибут "Можливість застосовування системи навігації" моделі ЯВ відображається у два атрибути зовнішньої моделі якості: "наявність головного навігаційного меню" і "наявність контекстного навігаційного меню". Опишемо один з них у вигляді шаблону (табл. 3.2).

Таблиця 3.2 – Опис атрибуту наявність головного навігаційного меню

Елемент шаблону	**Екземпляр елементу**
Характеристика	**Функціональність**
Підхарактеристика	**Навігація та перегляд інформації**
Назва атрибуту	**Головне навігаційне меню**
Визначення атрибуту	Навігаційне меню – це кнопки, іконки, посилання, що об’єднані в блок меню і служать для переміщення по сайту. Головне навігаційне меню – це меню, яке присутнє і незмінне на всіх сторінках сайту, доступних відвідувачу.
Шкала вимірювань	Порядкова (присутнє, відсутнє)
Процедура визначення, протокол, Х	Значення атрибуту можна визначити лише шляхом перегляду і визначення наявності чи відсутності головного меню. Цей атрибут не вказує на якість структури і оформлення меню, а лише на його наявність чи відсутність.
	Примітки
Тип збору даних та підрахунку	Ручний
Інтерпретація значення оцінки	Чим вище – тим краще
Метрика	Завершеність функціональної реалізації
Пріоритетність	0,8

Враховуючи особливості web-застосувань, зокрема за функціональним призначенням та взаємодією з користувачем, підхарактеристику "Функціональна повнота" характеристики "Функціональність" для зручності застосовування та чіткості формулювання вимог пропонується розділити на декілька підхарактеристик (рис. 3.1).

Рисунок 3.1 – Модифікована модель ЗЯ для області WWW

Прикладами зовнішніх атрибутів і метрик для такої характеристики якості як надійність, можуть бути середній час між відмовами, число усунутих дефектів при тестуванні, інтенсивність відмов та ін. Однак внутрішні, зовнішні й атрибути ЯВ взаємозалежні. Досягнення ЯВ залежить від задоволення атрибутів ЗЯ, які, у свою чергу, залежать від задоволення відповідних атрибутів ВЯ.

Виходячи з вищесказаного, для кінцевого застосовування побудованої моделі ЗЯ web-застосувань та іншого програмного забезпечення для Web, модель якості (рис. 3.1.) довизначаються атрибутами та метриками якості.

Слід зауважити, що побудована модель якості без значних модифікацій може бути застосована не лише при розробці вимог до web-застосувань, але й для оцінки більшості сайтів або іншого програмного забезпечення для WWW. Однак, атрибути, що відносяться до підхарактеристики "Функціональна повнота" характеристики "Функціональність", потрібно обов’язково переглянути відносно конкретної області застосування і вони, як правило, різні для різних web-застосувань. Так, наприклад, атрибути вищезгаданої підхарактеристики будуть абсолютно різні для web-системи дистанційного

навчання та будь-якого електронного магазину, що пояснюється зовсім різним призначенням цих систем.

Також для різних систем при оцінюванні їх якості потрібно переглянути пріоритетність визначену для атрибутів якості, оскільки вони вказують на рівень важливості даного атрибуту для конкретної предметної області. Для визначення пріоритету та рівня залежності атрибутів зовнішньої моделі якості використаємо метод SQFD.

Попередньо на базі простого алгоритму вибору визначимо пріоритети атрибутів моделі ЯВ. Для цього було проведено опитування (анкетування) експертів відносно переваги одного атрибута моделі ЯВ над іншим по кожній з характеристик $H_i^u,\ i \in N_u^k,\ K = \overline{1, S_i}$. При цьому вагові множники вибирались за транзитивною шкалою та базою 2.

Наведемо приклад для визначення пріоритету деяких атрибутів характеристики "Задоволеність" моделі ЯВ на базі простого алгоритму вибору. Для цього введемо наступні позначення:

H_1^U - характеристика "Задоволеність" моделі ЯВ;

$A_{ij}^U, i = 1, j = 1, N_i$ – атрибути характеристики "Задоволеність", зокрема

A_{11}^U – "Можливість застосовування системи навігації";

A_{12}^U – "Зрозумілість загальної структури сайту";

A_{13}^U – "Естетичне оформлення сайту";

A_{14}^U – "Можливість застосовування системи ведення відвідуваності сайту";

A_{15}^U – "Можливість застосовування системи для ведення статистики отриманих послуг";

A_{16}^U – "Наявність засобів наповнення інформаційного контенту".

Провівши статистичну обробку результатів анкетування, отримано наступні значення переваг одного атрибута над іншим: $\alpha_{12} = 1, \alpha_{23} = 1/2, \alpha_{34} = 8,\ \alpha_{45} = 1,\quad \alpha_{56} = 1/4$. Виходячи з рівності отримаємо

значення вектора пріоритетів $\alpha = (0{,}18;\ 0{,}18;\ 0{,}36;\ 0{,}045;\ 0{,}045;\ 0{,}18)$.

$\alpha_{ij} = \alpha_i / \alpha_j$,

Отримані значення пріоритету вимог ЯВ за характеристикою "Задоволеність" записують справа у "будинку якості". Виходячи із представлення вектора $\alpha = (0{,}18;\ 0{,}18;\ 0{,}36;\ 0{,}045;\ 0{,}045;\ 0{,}18)$, можна зробити висновок про те, що найбільш важливими для користувачів є атрибути, які відображають естетичні аспекти оформлення сайту, менш важливими – можливість застосовування системи навігації та інформаційного наповнення сайту, а також зрозумілість структури сайту. Практично не цікавлять користувачів властивості сайту, які відображають загальну його відвідуваність та ведення статистики отриманих послуг.

Частковий випадок представлення "будинку якості", який показує кореляцію та пріоритети між вимогами ЯВ та вимогами ЗЯ для області WWW, наведено у табл. 3. При цьому компоненти моделі ЗЯ позначено наступним чином:

H_1^x – характеристика "Функціональність" моделі ЗЯ.

$\ddot{I}_{11}^{x}$ – підхарактеристика "Навігація і перегляд інформації" характеристики "Функціональність", до якої входять наступні атрибути:

A_{111}^x – "Наявність контекстного меню";

A_{112}^x – "Наявність головного навігаційного меню";

$\ddot{I}_{12}^{x}$ – підхарактеристика "Керування інформаційним наповненням сайту" характеристики "Функціональність" до якої входять атрибути:

A_{121}^x – "Редагування стилів"

A_{122}^x – "Редагування сторінок та розділів сайту"

H_2^x – характеристика "Комфортність застосовування" моделі ЗЯ;

$\ddot{I}_{21}^{x}$ – підхарактеристика "Зрозумілість" характеристики "Комфортність застосовування", до складу якої входять наступні атрибути:

A_{211}^x – "Зрозумілість загальної структури сайту";

A^x_{212} – "Карта сайту";

$\ddot{I}^{\ x}_{22}$ – підхарактеристика "Комфортність навчання" характеристики "Комфортність застосовування", до складу якої входить атрибут A^x_{221} – "Питання, що часто задають".

$\ddot{I}^{\ x}_{23}$ – підхарактеристика "Комфортність роботи" характеристики "Комфортність застосовування", до складу якої входить атрибут A^x_{231} – "Комфортність і наочність навігації";

$\ddot{I}^{\ x}_{24}$ – підхарактеристика "Привабливість" характеристики "Комфортність застосовування", до складу якої входять наступні атрибути:

A^x_{241} – "Однорідність стильового оформлення";

A^x_{242} – "Групування головних елементів управління";

A^x_{243} – "Однорідність кольорового оформлення посилань";

A^x_{244} – "Сталість елементів керування";

A^x_{245} – "Підтримка різних мов";

H^x_3 – характеристика "Комфортність супроводу" моделі ЗЯ;

$\ddot{I}^{\ x}_{31}$ – підхарактеристика "Аналізованість" характеристики "Комфортність супроводу";

A^x_{311} – атрибут "Ведення лог-файлів" підхарактеристики "Аналізованість"

Таблиця 3.3 – Матриця залежностей та пріоритетів атрибутів ЯВ та ЗЯ

		H_1^x				H_2^x									H_3^x	Пріоритет вимог	Пріоритет атрибутів моделі якості у використанні	
		$\ddot{I}_{11}^x$		$\ddot{I}_{12}^x$		$\ddot{I}_{21}^x$		$\ddot{I}_{22}^x$	$\ddot{I}_{23}^x$	$\ddot{I}_{24}^x$					$\ddot{I}_{31}^x$			
		A_{111}^x	A_{112}^x	A_{121}^x	A_{122}^x	A_{211}^x	A_{212}^x	A_{221}^x	A_{231}^x	A_{241}^x	A_{242}^x	A_{243}^x	A_{244}^x	A_{245}^x	A_{311}^x			
H_1^U	A_{11}^U	6	9	0	0	3	3	0	9	0	6	0	0	0	0	0,18	45	16%
	A_{12}^U	0	6	0	0	9	9	6	9	3	3	0	6	6	0	0,18	72	23%
	A_{13}^U	0	0	0	0	0	0	0	0	9	6	9	9		0	0,45	39	4%
	A_{14}^U	0	0	0	0	0	0	0	0	0	0	0	0	0	9	0,45	9	1%
	A_{15}^U	0	0	0	0	0	0	0	0	0	0	0	0	0	9	0,37	69	44%
	A_{16}^U	0	0	9	9	0	0	0	0	0	0	0	0	0		0,18	33	12%
		Пріоритет атрибутів зовнішньої моделі якості																
		6	21	9	15	36	21	15	33	12	15	9	15	15	18			
		2%	9%	3%	5%	13 %	10 %	8%	14%	7%	6%	1%	8%	7%	7%			

Завдавши граничне значення пріоритету для атрибутів ЗЯ Pg = 15, отримаємо набір атрибутів, які включаються у вимоги ЗЯ.

$$\{A^x_{122}, A^x_{221}, A^x_{242}, A^x_{245}, A^x_{244}, A^x_{311}\} \quad (3.3)$$

3.5 Вибір ахрітектури web - застосування

На базі аналізу функціональності WEB - застосування та вимог ЗЯ виберемо альтренативні ахрітектури із каталога зразків проєктування [6].

Ахрітектура незалежних елементів складається з джерела даних і деякої кількості клієнтів, дані клієнтів обновляються, коли відбувається зміна в джерелі даних. Для цього можна використати зразок проєктування Observer. В якості альтренативи розглянемо також систему яка керується подіями, відповідну

ахрітектуру незалежних компонентів і зразок проєктування State. В даній ахрітектурі застосування представляють такими, що складаються з компонентів, кожнийз яких знаходиться в стані чекання, поки не відбудеться певна подія.

Третій зразок проєктування візьмемо з ахрітектури структурних патернів, а саме зразок Decorator. Це задача багатокритерійної оптимізації.

Для її рішення можна використати метод аналізу ієрархій Сааті. Але його застосування пов'язане з обчисленням власних чисел матриці попарних порівнянь а також може привести до некоректних результатів. Тому було використано простий метод розв'язання задачі [16].

Представимо схему методу на конкретному прикладі.

Розглянемо проблему визначення оптимальної ахрітектури з трьох альтренативних за сукупністю критеріїв якості (3).

Будемо розглядати наступні частинні критерії оптимальності, що характеризують в сукупності вихідний показник:

$f1 - A^x_{122}$

$f2 - A^x_{221}$

$f3 - A^x_{242}$

$f4 - A^x_{245}$

$f5 - A^x_{244}$

$f6 - A^x_{311}$

Передбачається, що всі введені частинні показники необхідно максимузувати, тобто більшому значенню кожного показника буде відповідати більш бажаний стан для замовника. Нехай маємо ситуацію, коли потрібно вибрати одну з трьох альтирнативних ахрітектур, позначених відповідно буквами:

- State – A;
- Observer – B;
- Decorator – C.

Будемо слідувати запропонованому простому алгоритму вибору з транзитивною шкалою і базою $a = 2$. Побудуємо вектор вагових множників для

сформульованих частинних критеріїв. Для цього експертами визначається значення коефіцієнтів переваги одного показника над іншим.

Будемо вважати, що по результатам експертної оцінки були отримані наступні дані:

$$\alpha_{12} = 2;\ \alpha_{23} = 4;\ \alpha_{34} = 1/4;\ \alpha_{45} = 1;\ \alpha_{56} = 4. \quad (3.3)$$

Тут рівність $\alpha_{12} = 2$, наприклад, значить, що частинний критерій *f*1 в два рази більш "важливий" ніж критерій *f*2 і т. д.

Скориставшись відношенням $\alpha_{ij} = \alpha_i / \alpha_j$ і за умови нормованості вектора α^1, безпосередньо отримуємо:

$$\alpha_1 = 0{,}364;\ \alpha_2 = 0{,}182;\ \alpha_3 = 0{,}045;\ \alpha_4 = 0{,}182;\ \alpha_5 = 0{,}182;\ \alpha_6 = 0{,}045. \quad (3.4)$$

Далі переходимо до процедури обчислювання значень частинних критеріїв оптимальності, що відповідають трьом варіантам A, B, C.

Спочатку, з допомогою того ж підходу, ранжуємо варіанти A, B, C по критерію f1. Нехай експерт вказав такі значення для кожного з коефіцієнтів досконалості:

$$\alpha^1_{12} = 1;\quad \alpha^1_{23} = 1/2. \quad (3.5)$$

Відповідний вектор вагових множників α^1 має компоненти:

0,250; 0,250; 0,500.

Вони інтерпретуються як значення функції f1 для трьох варіантів A, B, C:

$f_1(A) = 0{,}250;\ f_1(B) = 0{,}250;\ f_1(C) = 0{,}500.$

Аналогічно визначаємо значення наступних частинних критеріїв для варіантів A, B, C:

$$\alpha_{12}^2 = 2; \quad \alpha_{23}^2 = 2;$$

$$f_2(A) = 0{,}571; \; f_2(B) = 0{,}286; \; f_2(C) = 0{,}143$$

$$\alpha_{12}^3 = 1; \; \alpha_{23}^3 = 1;$$

$$f_3(A) = 0{,}333; \; f_3(B) = 0{,}333; \; f_3(C) = 0{,}333$$

$$\alpha_{12}^4 = 1/2; \; \alpha_{23}^4 = 1/4;$$

$$f_4(A) = 0{,}091; \; f_4(B) = 0{,}182; \; f_4(C) = 0{,}727.$$

$$\alpha_{12}^5 = 4; \; \alpha_{23}^5 = 2;$$

$$f_5(A) = 0{,}727; \; f_5(B) = 0{,}182; \; f_5(C) = 0{,}091.$$

$$\alpha_{12}^6 = 1/2; \; \alpha_{23}^6 = 2;$$

$$f_6(A) = 0{,}250; \; f_6(B) = 0{,}500; \; f_6(C) = 0{,}250.$$

Скориставшись методом лінійної згортки,

$$J(x_i) = \sum_{k=1}^{6} \alpha_k f_k(x_i), \tag{3.6}$$

отримаємо значення узагальненого критерію оптимальності для трьох варіантів $x_1 = A, x_2 = B, x_3 = C$:

Відповідно, до цього методу, найбільш оптимальною визначилася ахрітектура С. Однак бачимо, що вибір А теж є прийнятним.

Можна також замість лінійної згортки в якості узагальненого критерія використати згортку Джоффриона, засновану на комбінації лінійної і максимінної згортки.

Легко підрахувати, що в методі, який пропонується використати ,загальне число порівнянь об'єктів по важливості, що потрібно виконати, дорівнює:

$$N_1 = m - 1 + m\,(n - 1) = mn - 1, \tag{3.7}$$

де m – число частинних критеріїв, n – кількість альтренатив.

В стандартних процедурах Сааті число порівнянь дорівнює

$$N_2 = \frac{m^2 - m}{2} + m\frac{n^2 - n}{2}, \tag{3.8}$$

що може бути суттєво більше N1. Так для m=6, n=10 маємо: $N_1 = 59$, $N_2 = 285$.

Крім того, як уже зазначалося, при використанні запропонованого підходу нема необхідності розв'язувати лінійні системи і шукати власні числа матриць.

4 СПЕЦІАЛЬНА ЧАСТИНА

4.1 Розробка класів на основі стратегій як засіб забезпечення якості програмного забезпечення

У цьому розділі описуються поняття стратегії (policy) і класів стратегій (policy classes), що грають важливу роль в створенні бібліотек, що ефективно забезпечують повторне використання коду при розробці програмного забезпечення. Коротко кажучи, проектування, засноване на стратегіях, дозволяє створювати клас з складною поведінкою з множини маленьких класів (званих стратегіями), кожний з яких відповідає тільки за один функціональний або структурний аспект. Як випливає з її назви, стратегія встановлює інтерфейс, відповідний певній властивості. Стратегії можна реалізовувати по-різному, оскільки їх інтерфейси цілком знаходяться в компетенції програміста.

Змішуючи і підганяючи стратегії одну до одної, можна моделювати величезну кількість видів поведінки, використовуючи невеликий набір елементарних компонентів.

У цьому розділі описується призначення стратегій, розглядаються деталі їх проектування і приводяться поради, що дозволяють розкласти клас на стратегії.

4.2 Різноманітність методів розробки програмного забезпечення

Область проектування програмного забезпечення як ніяка інша технічна дисципліна демонструє велику різноманітність методів: одну і ту ж задачу можна правильно вирішити самими різними способами, причому між правильним і неправильним рішенням лежить нескінченна кількість нюансів. Кожен новий спосіб відкриває новий світ.

При виборі одного з рішень виникає безліч можливих варіантів, починаючи з рівня системної архітектури і закінчуючи щонайменшими деталями кодування. Таким чином, розробка програмних систем полягає у виборі рішень з гігантського числа варіантів.

Розглянемо простий приклад низькорівневої розробки – інтелектуальні вказівники. Інтелектуальні вказівники (smart pointers) – це класи, які можуть бути як одно- так і багатопотоковими. Вони можуть використовувати різні стратегії володіння ресурсами, а також різні компроміси між безпекою і швидкістю. Крім того, інтелектуальні вказівники можуть підтримувати або не підтримувати автоматичне перетворення в звичайні вказівники. Всі ці властивості можна вільно комбінувати, причому зазвичай в конкретній області, для якої призначено програмне забезпечення, якнайкращим є лише одне рішення.

Різноманітність методів створення програмного забезпечення постійно бентежить розробників-початківців. Перед ними стоїть конкретне завдання. Що застосувати для її успішного вирішення? Події? Об'єкти? Спостерігачі? Зворотні виклики? Віртуальні класи? Шаблони? Якщо абстрагуватися від конкретних деталей, різні рішення виявляться еквівалентними.

На відміну від новачка, досвідчений розробник програмного забезпечення знає, що працює, а що – ні. Для кожного конкретного завдання існує безліч методів рішення. Вони володіють своїми перевагами і недоліками, які визначають, наскільки той або інший метод підходить для вирішення поставленого завдання. Рішення, яке здавалося цілком прийнятним на папері, на практиці може виявитися помилкою.

Створювати програмне забезпечення складно, оскільки розробник постійно повинен робити вибір. А в програмуванні, як і в житті взагалі, важко зробити правильний вибір.

Досвідчений розробник знає, який вибір веде до поставленої мети, тоді як новачок кожного разу робить крок в невідомість. Досвідчений архітектор програмного забезпечення нагадує хорошого шахіста: він бачить на багато ходів вперед. Для цього потрібно довго вчитися. Можливо тому геніальні програмісти бувають дуже молодими, тоді як геніальні розробники програмного забезпечення зазвичай знаходяться в зрілому віці.

Окрім головоломок, що постійно виникають перед розробниками, що починають займатись створенням програм, комбінаторна природа архітектурних рішень доставляє багато клопоту творцям бібліотек. Для того, щоб реалізувати бібліотеку, придатну для створення програмного забезпечення, розробник повинен класифікувати і адаптувати безліч типових ситуацій. Бібліотеку слід залишити відкритою для модифікацій, щоб прикладний програміст міг пристосувати її для своїх потреб, створивши конкретну програму.

Дійсно, як упакувати гнучкі, грунтовно розроблені компоненти в бібліотеку? Як надати користувачеві можливість настроювати ці компоненти? Ось яким питанням присвячений цей розділ.

4.3 Недоліки універсального інтерфейсу

Реалізувати все під оболонкою універсального інтерфейсу – невдале рішення. Це пояснюється наступними причинами.

До основних негативних наслідків такого вибору відносяться інтелектуальні витрати, величезний розмір і неефективність бібліотеки. Гігантські класи дуже непродуктивні, оскільки на їх вивчення потрібно витрачати великі зусилля, вони дуже великі, а програми, що використовують такі класи, працюють набагато повільніше, ніж аналогічні програми, розроблені вручну.

Проте чи не найважливішою проблемою, зв'язаною з використанням універсального інтерфейсу, є втрата безпеки статичних типів (static type safety). Одна з основних цілей архітектури будь-якого програмного забезпечення – втілення деяких аксіом "за визначенням". Наприклад, не можна одночасно створювати два об'єкти певного класу. В ідеалі розробник повинен накладати більшість обмежень ще на етапі компіляції.

У великому всеосяжному інтерфейсі дуже важко врахувати всі подібні обмеження. Зазвичай конкретні обмеження семантично відповідають лише частині інтерфейсу. Це веде до збільшення розриву між синтаксичною і семантичною правильністю програм, що використовують дану бібліотеку.

Розглянемо, наприклад, аспект реалізації об'єкту класу, пов'язаний з безпекою роботи в багатопотоковому середовищі. Якщо бібліотека повністю інкапсулює потокові властивості, то користувач конкретної, непереносимої системи не може використовувати бібліотеку. Якщо ця бібліотека надає доступ до основних незахищених функцій, виникає небезпека, що програміст зруйнує бібліотеку, написавши код, який буде синтаксично правильним, але семантично невірним.

А що, коли бібліотека реалізовуватиме різні проектні рішення у вигляді різних класів скромнішого розміру? Кожен клас міг би утілювати конкретне проектне рішення. При використанні інтелектуальних вказівників, наприклад, природно чекати появи численних реалізацій: singietonThreadedSmartPtr, MultiThread-edsmartPtr, RefCountedSmartPtr, RefLinkedSmartPtr тощо.

Для цього підходу характерне різке зростання кількості різних варіантів проектних рішень. Чотири згадані класи можуть привести до появи нових комбінацій, наприклад, у вигляді класу SingieThreadedRefCountedSmartPtr. Перетворення типів приведе до ще більшої кількості комбінацій, які врешті-решт вийдуть за рамки можливостей як програміста, так і користувача бібліотеки. Очевидно, що йти цим шляхом не варто. Подолати експоненціальне зростання варіантів за допомогою грубої сили неможливо.

Такого роду бібліотеки не тільки вимагають для своєї розробки і застосування величезних розумових зусиль, але і є украй негнучкими. Щонайменша непередбачена настройка – наприклад, спроба ініціалізувати конкретним значенням інтелектуальні вказівники, створені за замовчуванням, – приводить всі ретельно розроблені класи бібліотеки в плачевний стан.

В процесі розробки програм на них накладаються певні обмеження. Отже, бібліотеки, орієнтовані на розробку програм, повинні давати користувачеві можливість формулювати свої власні обмеження, а не нав'язувати вбудовані. Раз і назавжди законсервовані проектні рішення можуть опинитися так само непридатними для бібліотек, орієнтованих на розробку програм, як, наприклад, константи, явно задані в звичайних програмах. Зрозуміло, набори "найбільш популярних" або "рекомендованих" готових рішень цілком допустимі, якщо програміст може змінювати їх в міру потреби.

Ці проблеми ілюструють несприятливу ситуацію, що склалася в області розробки бібліотек. Великої кількості низькорівневих універсальних і спеціалізованих бібліотек, що полегшують розробку програм, тобто структур високого рівня, практично немає. Це парадоксальна ситуація, оскільки будь-яке нетривіальне застосування утілює якісь проектні рішення. Отже, бібліотеки, орієнтовані на створення програм, повинні були б знайти застосування в більшості застосувань.

Цей пропуск спробували заповнити спеціальні середовища розробки (frameworks). Проте вони в основному призначені для того, щоб пов'язати додаток з конкретним проектним рішенням, а зовсім не для того, щоб допомогти користувачеві вибрати і налаштувати свій проект. Якщо програміст прагне реалізувати свій оригінальний проект, він повинен починати все "з нуля", створюючи класи, функції і т.п.

5 ОБҐРУНТУВАННЯ ЕКОНОМІЧНОЇ ЕФЕКТИВНОСТІ

Встановлення економічної доцільності проведення розробки, є основною метою виконання розділу організаційно-економічного обґрунтування дипломної роботи [20].

Метою роботи є розробка методики оцінювання якості програмного забезпечення.

5.1 Розрахунок норм часу на виконання науково-дослідної роботи

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

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

Основні етапи при виконанні НДР (розробки програми):

- Постановка проблеми.
- Аналіз виконаних досліджень.
- Аналіз і формалізація інформаційної системи.
- Проектування інформаційної системи.
- Обробка даних та розробка програми.
- Створення вихідних документів.
- Оформлення документації.

Для оцінки тривалості виконання окремих робіт використовують нормативи часу або попередній досвід. До таких нормативів відносять тривалість написання операції (команд), яка на деяких підприємствах становить:

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

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

при трьох оцінках:

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

при двох оцінках:

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

де $T_{вс}$ – очікуване (середнє) значення тривалості виконання етапу (стадії); t_{min}, $t_{н.й}$, t_{max} – відповідно мінімальна, найбільш імовірна і максимальна оцінки тривалості виконання етапу (стадії).

Для визначення загальної тривалості проведення НДР (розробки програми) доцільно дані витрат часу на виконання окремих стадій (етапів) звести у таблицю 5.1.

Середній час виконання НДР (розробки програми)

Таблиця 5.1 – Основні етапи і час їх виконання у НДР

Номер етапу	Середній час виконання стадії (етапу), год.	
	Інженер	Керівник
Постановка завдання	25	4
Аналіз виконаних досліджень	65	5
Аналіз і формалізація інформаційної системи	70	7
Проектування інформаційної системи	95	10
Обробка даних та розробка програми	108	11
Створення вихідних документів	100	6
Оформлення документації	63	5
Разом	526	48

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

5.2 Розрахунок витрат на проведення НДР

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

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

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

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

При системному розв’язанні питання про облік праці і заробітної плати велике значення має умовно-постійна (нормативна, довідкова та інша) інформація, яка в даному разі характеризує переважно постійних виконавців (людей і механізми) та постійні процеси (технологічні операції). Тому у першу чергу зміст по обліку праці і заробітної плати неодмінно повинна входити інформація про виконавців (облік складу працівників).

Для правильної оплати потрібні достовірний і своєчасний облік виробітку та визначення його якості. Облік виробітку і його оплати — найбільш трудомістка ділянка роботи, вона складається з ряду проміжних операцій.

Застосування ПЕОМ у вигляді АРМ, а також широке використання умовно-постійної інформації значно поліпшує облік показників про виробіток безпосередньо відноситься до обліку результатів і одночасно ці ж данні використовуються для нарахування відрядної заробітної плати. Так здійснюється зв'язок між цими ділянками обліку, а також взаємоконтроль цих показників.

Основна з/п складається із прямої з/п і доплати, яка при укрупнених розрахунках становить 25% – 35% від прямої з/п. При розрахунку з/п кількість робочих днів в місяці слід приймати — 25,4 дні/міс., що відповідає 203,2 год./міс. Приймемо розмір місячних окладів керівника 3600 грн. та інженерів 3420 грн. згідно існуючих на даний час норм.

Пряма з/п визначається:

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

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

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

Для інженера: ЗП = (3420·526)/203,2=6675,80 грн.;

Для керівника: ЗП = (3600·48)/203,2=677,95 грн.;

Величина доплат:

$$ЗП_i = ЗП \cdot К_i . \quad (5.4)$$

Тут $К_i$ – коефіцієнт доплат (0,25 – 0,35).

Вибираємо коефіцієнт 0,28.

Для інженера: $ЗП_i$=6675,80 *0,28=2029,25грн.

Для керівника: $ЗП_i$=677,95*0,28=205,86 грн.

Основна з/п: $ЗП = ЗП + ЗП_i$.

Для інженера: $ЗП_о$ = 6675,80 +1029,25=7705,05 грн.

Для керівника: $ЗП_о$ =677,95+105,86 =783,81 грн.

Величина додаткової з/п

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

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

Нехай $К_Д$=0,1, тоді:

$ЗП_Д$ = 7705,05 *0,1=770,5 грн.

$ЗП_Д$ =783,81 *0,1=78,38 грн.

для інженера і керівника відповідно.

Витрати, на проведення НДР (розробки програми) крім річного фонду заробітної плати включають ще й соціальні нарахування.

Всього норматив нарахувань на заробітну плату інженера становить

21% – (ЗПо+$ЗП_Д$)·0,38=5175,55*0,38=1966,71 грн.

Для керівника: 179,99*0,38=201,96 грн.

Результати обрахунків по з/п занесемо в таблицю 5.2.

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

Для розрахунку витрат на консультації, врахуємо, що консультації були надані в обсязі 10 год., вартість їх 50 грн.

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

Категорія працівників	Основна заробітна плата			Додаткова заробітна плата	Нарахування на заробітну плату	Всього витрати на заробітну плату
	Пряма заробітна	Доплати	Всього			
Інженер	3675,80	1029,25	4705,05	470,5	1966,71	7142,26
Керівник	377,95	105,86	483,81	48,38	201,96	734,15
Всього	4053,75	1135,11	5188,86	518,88	593,07	7876,41

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

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

де $М_з$ – затрати на матеріали;

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

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

n – кількість найменувань матеріалів.

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

Результати розрахунку затрат на матеріали зводяться в таблицю 5.3.

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

Таблиця 5.3 – Визначення величини затрат на матеріали

Найменування матеріальних ресурсів	Одиниця виміру	Норма витрат	Ціна за одиницю, грн.	Затрати матеріалів, грн.	Транспортно-заготівельні	Загальна сума витрат на матеріали, грн.
1. Основні матеріали						
Комп’ютер	штука	3	5000	15000	900,00	15900,00
Принтер	штука	2	1000	2000	120,00	2120,00
Монітор	штука	3	1500	4500	270,00	4770,00
2. Допоміжні матеріали						
Папір	Упаковка	3	35	105	6,30	111,30
Інсталяційні диски	Штук	10	5	50	3,00	53,00
Разом						22954,30

Загально виробничі витрати при укрупнених розрахунках приймаємо на рівні 70% – 90% від суми основної і додаткової з/п інженерів, яка була нарахована за роботу по проведенні НДР (розробки програмного продукту). Аналогічно визначаються адміністративні витрати, які доцільно приймати на рівні 50% – 60% від суми основної і додаткової з/п інженерів. Позавиробничі витрати слід приймати на рівні 3% – 7% від виробничої собівартості.

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

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

Таблиця 5.4 – Калькуляція собівартості проведення НДР (розробки програми)

Статті витрат	Витрати, грн.	В відсотках до загальної суми
1 . Основна заробітна плата	5188,90	14,07
2. Додаткова заробітна плата	518,88	1,41
3. Нарахування на заробітну плату	593,07	1,61
4. Консультації	50,00	0,14
5. Матеріали	22954,00	62,25
6. Експериментально-виробничі витрати	570,00	1,55
7. Загально виробничі витрати	3625,00	9,83
Разом виробнича собівартість	33500,00	90,85
8. Адміністративні витрати	2590,00	7,02
9. Позавиробничі витрати	785,00	2,13
Повна собівартість	36875,00	100

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

- облік складу і рухів працівників при розробці програмного продукту;
- табельний облік робітників і службовців, а також облік використання робочого часу у різних аспектах.

5.3 Розрахунок ціни НДР і економічна ефективність від використання програми

Ціну НДР (розробки програми) можна визначити:

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

де $C_{пр}$ – собівартість НДР (розробки програми), грн.;

N –кількість замовлень, од.;

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

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

Ц=(36875,00/1+450)+5750=43075,00 грн.

Економічна ефективність від використання НДР зумовлена:

– скороченням трудовитрат при виконанні певних завдань;

– скороченням машинного часу при виконанні певних завдань.

При визначенні економічної ефективності необхідно порівняти використовуваний (базовий) програмний продукт і пропонований. З допомогою відповідних розрахунків (в разі значної складності використання експертних оцінок) визначається скорочення трудовитрат і (або) машинного часу, і як наслідок – економія коштів при використанні нового програмного продукту.

Для визначення ефетивності продукту розраховують чисту приведену цінність NPV і термін окупності $Т_{ок}$:

$$NPV = \sum((Д_t - B_t)/(1+i)^t), \quad (5.8)$$

де $Д_t$, – повний дохід за рік t при використанні програми ; B_t – повні витрати за рік t при використанні програми; t – відповідний рік проекту; i – дисконтна ставка (0,3).

Нехай повний дохід за рік при використанні програми 10000 грн., тоді чиста приведена цінність:

$$NPV = ((10000-600)/(1+0,3)^1) + (10000-600)/(1+0,3)^2) = 12792,90,$$

Термін окупності визначається за формулою:

$$Т_{ок} = Ц/\sum(Д_t/(1+i)^t), \quad (5.9)$$

$Т_{ок}$=43075,00/(10000/(1+0,3)^1+10000/(1+0,3)^2) ≈ 3,4 роки.

Таблиця 5.5 – Основні показники ефективності

№ п/п	Назва показника	Один, вимір.	Величина
1	Витрати часу на розробку програми	год.	574
2	Витрати на розробку програми	грн.	36875,00
3	Кількість покупців програми	од.	1
4	Ціна програми	грн.	43075,00
5	Чиста приведена цінність програми	грн.	12792,90
6	Термін окупності витрат по проекту	рік	3,4

Отже собівартість створення програми 36875,00 грн. Розробка окупиться за 3,4 роки.

6 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

6.1 Загальні вимоги законодавства з охорони праці в галузі інформаційних технологій

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

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

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

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

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

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

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

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

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

6.2 Розрахунок захисного заземлення

Відповідно до ГОСТ 12.1.030-81 захисне заземлення повинне забезпечити захист людей від поразки електричним струмом, при дотику до металевих неструмоведучих частин, які можуть виявитися під напругою. Заземленням називається навмисне з'єднання електроустановок із заземлюючим пристроєм, Заземлювачем називається провідник, що перебуває в зіткненні із землею або її еквівалентом. Заземлюючим провідником називається провідник, що з'єднує заземлені частини із заземлювачем. Сукупність з'єднуючих провідників і заземлювачів називається заземлюючим пристроєм. Для установок потужністю

не більше 100 кВт опір заземлюючого пристрою не повинне перевищувати 10 Ом, для установок потужністю більше 100 кВт – 4 Ом.

Розрахунок штучного заземлювального пристрою при відсутності природних заземлювачів.

Вихідні дані:

Захищуваний об'єкт – комп'ютерна мережа.

Захищуваний об'єкт – стаціонарний.

Напруга мережі – 220 В.

Виконання мережі – з глухозаземленою нейтраллю.

Тип заземлювального пристрою – вертикальні труби.

Розміри вертикальних заземлювачів:

Довжина – 6 м.

Діаметр труби – 0,60 м.

Товщина стінки труби – 0,06 м.

Висота труби – 0,6 м.

Відношення відстані між трубами до їхньої довжини:

$$\frac{L_B}{l_B} = 1 \qquad (6.1)$$

Розмір горизонтального заземлювача (з'єднувальної стрічки): довжина $L_Г = L_{З.С.}$ – згідно з розрахунком, м; ширина горизонтальної з'єднувальної стрічки b_c=0,04 м.

Глибина закладання вертикальних заземлювачів h_B=0,6.

Розміщення заземлювачів попередньо приймають за чотирикутним контуром при числі стержнів від 4 до 100 та в один ряд при числі стержнів від 2 до 20.

Грунт – супісок; склад – однорідний; вологість – мала; агресивність – нормальна.

Кліматична зона – II.

Розрахунок:

1. Визначаємо характеристику навколишнього середовища в приміщенні організації: за пожежною небезпекою згідно з ПУЕ воно відноситься до класу П-II; за вибухонебезпекою згідно з ПУЕ – до класу В-I; за ступенем ураження електричним струмом – без підвищеної та особливої небезпеки.

2. Визначаємо $R_Д$ – допустиме (нормативне) значення опору розтікання струму в заземлювальному пристрої, $R_Д \leq 4$ *Ом*.

3. Обраховуємо $K_{С.В.}$ – приблизне значення питомого опору ґрунту, що рекомендується для розрахунку – $\rho_{ТАБЛ} = 300$ *Ом* $\cdot$ *м*.

4. Визначаємо $K_{С.В.}$ – коефіцієнт сезонності для вертикальних заземлювачів для денної кліматичної зони. За довідковою інформацією приймаємо $K_{С.В.}$=1,5.

5. Обраховуємо значення $K_{С.Г.}$ – коефіцієнт сезонності для горизонтального заземлювача згідно з кліматичною зоною. За довідковою інформацією приймаємо ; $K_{С.Г.}$=3,5.

6. Визначаємо $\rho_{РОЗР.В.}$ – розрахунковий питомий опір ґрунту для вертикальних заземлювачів:

$$\rho_{РОЗР.В.} = \rho_{ТАБЛ} \cdot K_{С.В.} = 300 \cdot 1{,}5 = 450 \text{ Ом} \cdot \text{м}. \qquad (6.2)$$

7. Розраховуємо $\rho_{РОЗР.Г.}$ – розрахунковий питомий опір ґрунту для горизонтальних заземлювачів:

$$\rho_{РОЗР.Г.} = \rho_{ТАБЛ} \cdot K_{С.Г.} = 300 \cdot 3{,}5 = 1050 \text{ Ом} \cdot \text{м}. \qquad (6.3)$$

8. Обраховуємо t – відстань від поверхні землі до середини вертикального заземлювача:

$$t = h_В + \frac{l_В}{2} = 0{,}6 + \frac{6}{2} = 3{,}6 \text{ м}. \qquad (6.4)$$

9. Визначаємо R_B – опір, Ом, розтікання струму в одному вертикальному заземлювачі:

$$R_B = \frac{\rho_{РОЗ.В}}{2\pi l_B} \cdot \left(ln\frac{2l_B}{d} + 0{,}5 \cdot ln\frac{4t + l_B}{4t - l_B} \right) = \\ = \frac{450}{2\pi \cdot 6} \cdot \left(ln\frac{2 \cdot 6}{0{,}60} + 0{,}5 \cdot ln\frac{4 \cdot 3{,}6 + 6}{4 \cdot 3{,}6 - 6} \right) = 41{,}05\ Ом \tag{6.5}$$

10. Визначаємо $n_{т.в.}$ – теоретична кількість вертикальних заземлювачів без врахування коефіцієнта використання $\eta_{в.в.}$, тобто $\eta_{в.в.}=1$:

$$n_{Т.В.} = \frac{R_B}{R_Д \cdot \eta_{В.В}} = \frac{41{,}05}{4 \cdot 1} = 10{,}26\ шт. \tag{6.6}$$

11. Визначаємо $\eta_{в.в}$ – коефіцієнт використання вертикальних заземлювачів при розташуванні їх згідно з вихідними даними або за чотирикутним контуром при числі заземлення, $n_{т.в.}=11$ та при відношенні $\frac{L_B}{l_B} = 1$. За довідником приймаємо $\eta_{в.в} = 0{,}42$.

12. Визначаємо $n_{н.в}$ – необхідна кількість штук, вертикально однакових заземлювачів з врахування коефіцієнта використання:

$$n_{Н.В} = \frac{R_B}{R_Д \cdot \eta_{В.В}} = \frac{41{,}05}{4 \cdot 0{,}42} = \frac{41{,}05}{1{,}68} = 24{,}43\ шт. \tag{6.7}$$

13. Визначаємо $R_{розр.в}$ – вертикальний опір, Ом, розтіканню струму у вертикальному заземленні при $n_{н.в} = 24{,}43$ без врахування з'єднувальної стрічки:

$$R_{РОЗР.В} = \frac{R_B}{n_{Н.В} \cdot \eta_{В.В}} = \frac{41{,}05}{10{,}26} = 4\ Ом. \tag{6.8}$$

14. Визначаємо $L_в$ – відстань між вертикальним заземлювачами за відношенням $\frac{L_B}{l_B}=1$, звідси

$$L_B = 1 \cdot l_B = 1 \cdot 6 = 6 \text{ м}. \quad (6.9)$$

15. Визначаємо $L_{з.с}$ – довжину, м, з'єднання стрічки горизонтального заземлювача:

$$L_{З.С} = 1{,}05 \cdot L_B (n_{Н.В} - 1) = 1{,}05 \cdot 6(24{,}43 - 1) = 147{,}60 \text{ м}. \quad (6.10)$$

16. Визначаємо $R_{г.з.с}$ – опір, Ом, розтікання струму в горизонтальному заземлювачі (з'єднувальній стрічці):

$$R_{Г.З.С} = \frac{\rho_{РОЗ.Г}}{2 \cdot \pi \cdot L_{З.С}} \cdot ln \frac{2 \cdot L_{З.С}^{\,2}}{2 \cdot b \cdot t} = \frac{1050}{2 \cdot 3{,}14 \cdot 147{,}61} \cdot ln \frac{2 \cdot (147{,}61)^2}{2 \cdot 0{,}04 \cdot 3{,}6} = 13{,}50 \text{ Ом}. \quad (6.11)$$

17. Визначаємо $\eta_{в.г}$ – коефіцієнт використання горизонтального заземлення при розташуванні вертикальних заземлювачів згідно з вихідними даними або за чотирикутним контуром при відношенні $\frac{L_B}{l_B}=1$ та необхідній кількості вертикальних заземлювачів $n_{н.в}$ = 24,43.

За довжину приймаю $\eta_{в.г}$ = 0,19.

18. Визначаємо $R_{розр.г}$ – розрахунковий опір, Ом, розтікання струму в горизонтальному заземленні (з'єднувальній стрічці) при числі електродів $n_г$= 1:

$$R_{РОЗР.Г} = \frac{R_{Г.З.С}}{n_Г \cdot \eta_{В.Г}} = \frac{13{,}5}{1 \cdot 0{,}19} = 71{,}05 \text{ Ом}. \quad (6.12)$$

19. Визначаємо $R_{розр.в.г}$ – розрахунковий теоретичний опір, Ом, розтікання струму у вертикальному та горизонтальному заземленні:

$$R_{РОЗР.В.Г} = \frac{1}{\frac{1}{R_{РОЗ.В}} + \frac{1}{R_{РОЗ.Г}}} = \frac{1}{\frac{1}{4} + \frac{1}{71,05}} = 3,78 Ом . \qquad (6.13)$$

20. Вибираємо матеріал та поперечний перетин з'єднувальних провідників. За довідковою інформацією вибираю голі мідні $S_м$ = 4 мм^2 провідники.

21. Вибираємо матеріал та поперечний перетин магістральної шини. За довідковою інформацією обираємо сталеву шину товщиною $\delta_с$ = 4мм і перетином не менше δ = 100 мм^2.

Схема з'єднання обладнання з магістральною шиною та з'єднання магістральної шини з заземлювальним пристроєм (рисунок 6.1).

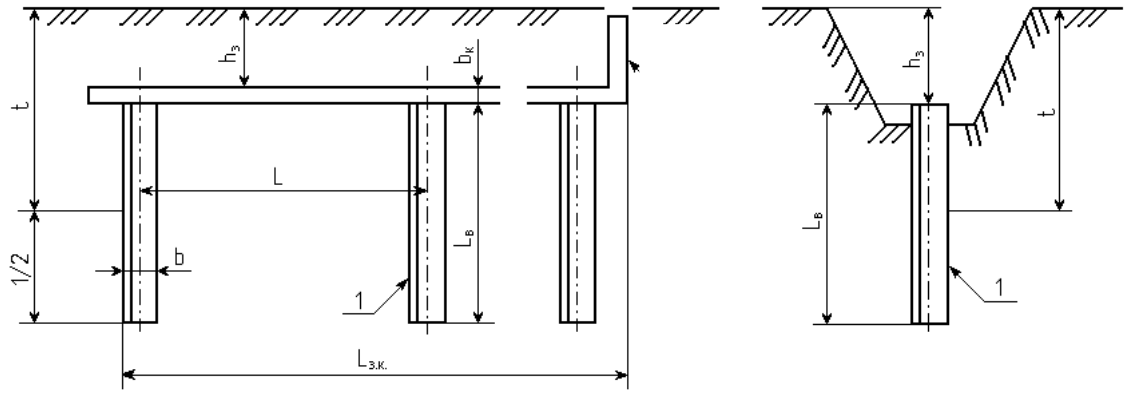

Рисунок 6.1 – Схема заземлювального контуру:

1 – вертикальний заземлювач; 2 – горизонтальний заземлювач; $h_з$ – глибина закладання заземлювачів; L – відстань між заземлювачами; $b_К$ – ширина квадрата; t – відстань від середини заземлювача до поверхні грунту; $L_{З,К}$ довжина горизонтального заземлювача; d – ширина кутника; $L_{Â}$ – довжина вертикального заземлювача.

6.3 Поняття та об’єкт аналізу технічної безпеки

Безпеку визначають як стан діяльності людини, за яким з визначе-ною ймовірністю виключено прояв небезпек або ж відсутня надзвичай-на небезпека. Безпека праці – це стан умов праці людини, за яких відсу-тня дія небезпечних і шкідливих факторів.

Об’єктом аналізу безпеки праці є виробнича система "людина – машина – навколишнє середовище" (ЛМС), в якій в єдиній комплекс, створений для виконання певних функцій, поєднані технічні об’єкти, люди і навколишнє середовище, які взаємодіють між собою.

Основними компонентами виробничої системи є людина, машина, навколишнє середовище, взаємодія між якими має ґрунтуватись на дотри-манні відповідних правил, нормативних документів і бути керованою.

Система ЛМС є багаторівневою за ієрархією управління. Ієрархія поділяє людей на особу, яка формує завдання, організовує й управляє вироб-ництвом, й особу, яка разом з технікою безпосередньо виконує це завдан-ня. Таким чином, людина системи ЛМС більш високого рівня розглядає людину і техніку системи ЛМС більш низького рівня як єдиний компонент – своєрідну людину-машину, призначену для здійснення замислу.

Крім рівнів і компонентів в системі ЛМС доцільно виділити окремі стадії її життєвого циклу:

1. Стадія проектування (визначення завдань, формування вимог, розрахунок параметрів).
2. Стадія реалізації (коли у процесі виробництва перша стадія реа-лізується на практиці).

3. Стадія експлуатації (коли система ЛМС здійснює покладені на неї робочі функції).

Вірогідність нещасного випадку зростає, як тільки людина попадає в поле дії небезпечного або шкідливого фактору. Це небезпечні зони, що характеризуються певним видом небезпеки, її інтенсивністю, часом і простором дії.

Таким чином, з точки зору аналізу й управління небезпеками необ-хідно розглядати та аналізувати структурні елементи системи ЛМС – рівні (вищий і нижчий), компоненти і стадії життєвого циклу.

Взаємодія компонентів, що входять до системи ЛМС, може бути штат-ною і нештатною. Нештатна взаємодія може виявлятися у вигляді надзви-чайної події – небажаних, незапланованих випадків, що порушують технологічний процес у відносно короткий відрізок часу. Відмова й інцидент, як пра-вило, передують надзвичайній події, але можуть мати і самостійне значення. До головних моментів аналізу небезпек належить пошук відповідей на такі питання:

1. Які об’єкти є небезпечними;
2. Яким надзвичайним подіям можна запобігти;
3. Які надзвичайні події неможливо усунути і як часто вони мати-муть місце;
4. Яку шкоду не усунуті надзвичайні події можуть спричинити людям, об’єктам, навколишньому середовищу.

Пошук причин надзвичайних подій призводить до аналізу системи управління небезпеками (СУН) на виробництві. Ці системи обов’язково включають такі компоненти, як наявність інформації, зворотних зв’язків та алгоритми функціонування.

Наявність зворотних зв’язків й інформаційної системи дозволяє про-водити збір даних щодо відхилень, відмов, проводити аналіз небезпек, по-рівнювати наслідки функціонування системи ЛМС з програмою управлін-ня небезпеками, приймати рішення. У виробничій системі ЛМС інформаційні

функції виконують: рапорти інспекторів, акти розслідування нещасних випадків, аварій, протоколи атестації робочих місць тощо.

6.4 Дії населення в надзвичайних ситуаціях (пожежа)

Якщо виникла пожежа, відлік часу йде на секунди, тому необхідно заздалегідь знати, де і які засоби пожежогасіння розміщуються та як ними користуватися.

Під час пожежі остерігайтеся: високої температури, задимленості та загазованості, обвалу конструкцій будинків і споруд, вибухів технологічного обладнання і приладів, падіння обгорілих дерев і провалів. Небезпечно входити в зону задимлення.

Заходи щодо рятування потерпілих з будинків, які горять, та під час гасіння пожежі:

– перед тим, як увійти у приміщення, що горить, накрийтесь мокрою ковдрою, будь-яким одягом чи щільною тканиною;

– відчиняйте обережно двері в задимлене приміщення, щоб уникнути спалахування від великого притоку свіжого повітря;

– в дуже задимленому приміщенні рухайтесь поповзом або пригинаючись; для захисту від чадного газу необхідно дихати через зволожену тканину; насамперед рятуйте дітей, інвалідів та старих людей;

– пам’ятайте, що маленькі діти від страху часто ховаються під ліжко, в шафу та забиваються у куток;

– виходити із осередку пожежі необхідно в той бік, звідки віє вітер;

– побачивши людину, на якій горить одяг, зваліть її на землю та швидко накиньте пальто, плащ або будь-яку ковдру чи покривало (бажано зволожену) і щільно притисніть до тіла, за необхідності викличте медичну допомогу;

– якщо загорівся ваш одяг, падайте на землю і перевертайтесь, щоб збити полум'я, ні в якому разі не біжіть – це ще більше роздмухує вогонь;

– під час гасіння пожежі використовуйте вогнегасники, пожежні гідранти, а також воду, пісок, землю, кошму, ковдри та інші засоби, пристосовані для гасіння вогню;

– бензин, гас, органічні масла та розчинники, що загорілися, гасіть тільки з допомогою пристосованих видів вогнегасників, засипайте піском або ґрунтом, а якщо осередок пожежі невеликий, накрийте його азбестовим чи брезентовим покривалом, зволоженою тканиною чи одягом;

– якщо горить електричне обладнання або проводка, вимкніть рубильник, вимикач або електричні пробки, а потім починайте гасити вогонь.

Дії, якщо пожежа застала у приміщенні:

– ви прокинулись від шуму пожежі і запаху диму, не сідайте в ліжку, а скотіться з нього на підлогу;

– повзіть підлогою під хмарою диму до дверей вашого приміщення, але не відчиняйте їх відразу;

– обережно доторкніться до дверей тильним боком долоні, якщо двері не гарячі, то обережно відчиніть їх та швидко виходьте;

– якщо двері гарячі – не відчиняйте їх, дим та полум'я не дозволять вам вийти;

– щільно зачиніть двері, а всі щілини і отвори заткніть будь- якою тканиною, щоб уникнути подальшого проникнення диму, та повертайтесь поповзом у глибину приміщення і вживайте заходи для порятунку;

– присядьте, глибоко вдихніть повітря, відчиніть вікно, висуньтеся та кричіть: "Допоможіть, пожежа!";

– ви не в змозі відчинити вікно – розбийте віконне скло твердим предметом та зверніть увагу людей, які можуть викликати пожежну команду;

– якщо ви вибрались через двері, зачиніть їх і поповзом рухайтесь до виходу із приміщення;

– обов'язково зачиніть за собою всі двері;

– під час пожежі заборонено користуватися ліфтами;

– якщо ви перебуваєте у висотному будинку, не біжіть вниз крізь вогнище, а користуйтеся можливістю врятуватися на даху будівлі.

У всіх випадках, якщо ви маєте змогу, зателефонуйте "01" і викличте пожежну команду.

Пожежі в лісах, стелах та на торфовищах.

Такі масові пожежі можуть виникати в спеку та при посухах від ударів блискавки, необережного поводження з вогнем, очищення поверхні землі випалюванням сухої трави та з інших причин. Вони можуть викликати загорання будівель в населених пунктах, дерев'яних мостів, дерев'яних стовпів ліній електромереж та зв'язку, складів нафтопродуктів та інших матеріалів, що горять, а також ураження людей та тварин.

Дії, якщо ви опинилися в осередку пожежі:

– не панікуйте та не приймайте поспішних, необдуманих рішень;

– не тікайте від полум'я, що швидко наближається, у протилежний від вогню бік, а рухайтесь крайкою полум'я проти вітру, закривши голову і обличчя одягом;

– з небезпечної зони, до якої наближається полум'я, виходьте швидко, перпендикулярно напряму поширення вогню;

– якщо втекти від пожежі неможливо, то вийдіть на відкриту місцевість або галявину, ввійдіть у водойму або накрийтесь мокрим одягом і дихайте повітрям, що над самою поверхнею землі, – воно тут менш задимлене, рот і ніс при цьому прикривайте одягом чи шматком будь-якої тканини;

– гасити полум'я невеликих низових пожеж можна, забиваючи його гілками листяних порід дерев, заливаючи водою, закидаючи вологим ґрунтом та затоптуючи ногами;

– під час гасіння пожежі не відходьте далеко від доріг та просік, не упускайте з виду інших учасників гасіння пожежі, підтримуйте з ними зв'язок з допомогою голосу;

– будьте обережні в місцях горіння високих дерев, вони можуть упасти та травмувати вас;

– особливо будьте обережні у місцях торф'яних пожеж, ввжайте, що там можуть створюватися глибокі вирви, тому рухайтеся, за можливості, перевіряючи палицею глибину шару, що вигорів;

– після виходу із осередку пожежі повідомте місцеву адміністрацію та пожежну службу про місце, розміри та характер пожежі.

Якщо людина знає правила поводження під час пожежі, то вона в змозі не лише вистояти за будь-яких обставин і врятувати своє життя, а й надати допомогу в рятуванні інших людей та врятувати матеріальні цінності від вогню.

7 ЕКОЛОГІЯ

7.1 Електромагнітне забруднення довкілля, його вплив на людину. Шляхи його зменшення

Несприятливий вплив на організм людини мають і електромагнітні випромінювання промислової частоти (50 Гц) та частот радіохвильового діапазону. У помешканнях електромагнітні поля створюють: радіоапаратура, телевізори, холодильники тощо, що має певну небезпеку. Справа в тому, що кожен внутрішній орган працює на певній частоті, наприклад, серце – біля 700 Гц (коливань в секунду), мозок у стані сну – 10 Гц, бадьорості – 50 Гц ін. Якщо поруч знаходиться постійне джерело електромагнітного випромінювання, яке працює на аналогічній (чи є кратною) частоті, що може призвести до збільшення або зменшення нормальної частоти роботи органу. Наслідком цього може бути головний біль, порушення сну, перевтома, навіть загроза виникнення стенокардії. Найбільш небезпечне випромінювання, коли людина (а особливо дитина) спить.

Безперечно, обійтися без електропобутових приладів неможливо, та й не потрібно. Головне – дотримуватись певних правил:

– у спальні не варто встановлювати комп’ютер, “базу” для радіотелефону, а також вмикати на ніч пристрої для підзарядки батарейок та акумуляторів;

– телевізор, музичний центр, відеомагнітофон на ніч треба вимикати з електромережі;

– електронний будильник не повинен стояти в узголів’ї;

– потужність мікрохвильових печей може змінюватись, тому час від часу треба звертатися до майстра, щоб контролювати рівень випромінювання.

7.2 Аналіз сучасних програмних продуктів опрацювання великих масивів екологічної інформації

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

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

Використання згаданих пакетів програм дає змогу автоматизувати процес статистичного дослідження в таких напрямах: створення файлів даних і таблиць; групування даних; графічний аналіз даних; розрахунок варіаційних характеристик вибіркових сукупностей; побудова рядів розподілу; аналіз рядів динаміки і прогнозування їх майбутніх рівнів; кореляційно-регресійний аналіз; багатомірний аналіз.

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

Передусім це стандарти користувацького інтерфейсу — MDI, використання буфера-обміну, механізму динамічного зв'язку (DDE) з іншими

додатками; система підтримує всі операції, реалізовані за допомогою методу Drag-and-Drop — «Перетягти та опустити», включаючи автозаповнення, інші.

Складніші процедури обробки даних у системі Stratgraphics виконує спеціалізований модуль Data Management — «Управління даними», а для обробки великих масивів даних або даних з довгими текстовими значеннями застосовують процедури Megafile Manager Data — «Менеджера мегафайлів».

Система Stratgraphics працює з чотирма типами документів. Це: електронна таблиця Spreadsheet, призначена для введення і перетворення первинних даних; електронна таблиця Scrollsheet — для виведення результатів аналізу; графік — для візуалізації результатів обробки та аналізу даних; звіт — файл у формі RTF (розширений текстовий формат), в якому зберігається текстова, числова і графічна інформація.

Усі статистичні процедури системи розбито на окремі модулі, кожен з яких об'єднує групу логічно зв'язаних між собою статистичних методів і в рамках конкретної моделі забезпечує повний і всебічний аналіз закономірностей.

У системі Statistica реалізовано принцип постійного логічного підказування. Якщо користувач не може визначитися щодо наступного кроку діалогу, через команду Enter система сама спрямує до відповідного діалогового вікна. Якщо виникають складнощі з вибором параметрів обчислювальної процедури, вони задаються системою «за умовчуванням».

Важливою характеристикою системи с наявність засобів всебічної графічної підтримки процесу обробки даних і візуалізації результатів аналізу. Графічні можливості й засоби системи унікальні. Вона включає сотні різних типів користувацьких і спеціальних статистичних графіків, доступних у будь-якому модулі й на будь-якому етапі статистичної обробки даних. Інструменти компонування складної графічної інформації з текстовою і числовою інформацією розглядаються у кожному модулі.

Використання сучасних комп'ютерних технологій обробки даних, інтерактивний спосіб взаємодії з системою перетворюють статистичний аналіз, моделювання та прогнозування в захоплююче дослідження закономірностей навколишнього світу. Завдяки різноманітним формам організації діалогу, максимально простій із звичними для статистики термінами мові спілкування,

наявності контекстно-залежної довідкової системи, мові програмування STATISTICA BASIC пакет є ефективним інструментом проведення статистичного дослідження як для користувача-початківця, так і для професіонала.

7.3 Проблема екологічності інформаційних і телекомунікаційних технологій

Екологія та технологія протягом півтора століть – від виникнення поняття екології в сучасному розумінні – залишались антагоністами. Вони не просто протистояли одне одному – існування одного виключало можливість існування другого. Але якщо перший етап розвитку обох явищ був боротьбою протилежностей, то після виходу і екології, і технології на якісно новий рівень відбулося єднання цих протилежностей.

Коли технології стали інформаційними (та телекомунікаційними), вони змогли сприйняти цілі й цінності екології. А екологія не просто отримала потужного союзника у перетворенні людського суспільства – вона набула реального, практичного змісту, перестала бути сухою філософською теорією, стала більш чи менш усвідомлюваним фоном повсякденного життя для більшості людства.

Інформаційні та телекомунікаційні технології, включивши в себе екологію в якості гуманних підвалин розвитку, перетворились на ідею Інформаційного суспільства, стали способом життя людства, запорукою нового циклу розвитку цивілізації та планети.

Екологія ж віднайшла спосіб втілення і розв’язання тисячолітнього конфлікту “людське проти природного”. В новому судженні зв’язок між двома елементами змінив характер: “або” перетворено Інформаційним суспільством на “та”.

На сьогодні основними практичними проблемами є екологічність інформаційних і телекомунікаційних технологій та технологічність (перш за все “інформаційність”) екологічних потреб. Якою мірою враховано інтереси людини, природи та планети у новітніх технологічних розробках? Якою мірою

екологічні вимоги можуть бути практично втілені за допомогою цих технологій (але перш за все – доведені до відома мешканців Землі)?

Зупинимось на першому аспекті. Останні дані дозволяють нам зробити висновок якщо не про абсолютну екологічну ефективність інформаційно-телекомунікаційних технологій, то про їх чітку екологічну спрямованість. Так найпотужніші комп'ютери світу працюють на екологічні програми: на сьогоднішній день найбільш потужним у світі суперкомп'ютером визнано IBM ASCI White. Його встановлено в американській урядовій дослідницькій лабораторії Lawrence Livermore National Laboratory й використовувано для створення повноцінної тривимірної моделі термоядерної реакції.

Це означає, що більше не треба здійснювати вибухи або запускати ненадійні – бо експериментальні – реакторні установки. Зникає ще одна небезпека для людини і природи: не забруднюватиметься атмосфера і ґрунт при видобуванні радіоактивних елементів та захороненні відпрацьованих – а отже менше хворітимуть мешканці відповідних районів та працівники, задіяні в цьому.

Другим за потужністю визнано комп'ютер, встановлений у дослідницькому центрі National Energy Research Scientific Computing Center (NERSC). Суперкомп'ютер центру NERSC окрім того визнано першим за потужністю серед систем, відкритих для загального користування. Ним користуються 2 тисячі різних дослідників, що займаються розробками у галузі створення екологічно чистих і більш економічних видів палива, вивченням глобальних змін клімату планети та іншими проблемами.

Використання інформаційних технологій для моніторингу екологічних систем та моделювання їх розвитку уможливило й інші серйозні міжнародні проекти. Так Організація об'єднаних націй має намір провести вивчення екологічного стану Землі. Згідно з офіційною заявою, зробленою представником Генерального секретаря ООН Фредом Екхартом, програма розрахована на чотири роки. До реалізації настільки масштабного проекту буде залучено більш як півтори тисячі вчених.

В результаті фахівці-екологи повинні дати оцінку теперішньому станові лісів, лук та полів Землі, а також прісних та солоних водойм. Передбачається, що

“обнародувані до 2005 року експертами висновки й рекомендації буде використано урядами різних країн для ухвалення більш компетентних та обґрунтованих рішень”, заявив Ф.Екхарт.

Загалом, концепція глобальних та локальних електронних інформаційних мереж є екологічною за своєю суттю. Це твердження вірне з точки зору як екології планети, так і екології людини, людської спільноти.

Інтернет оберігає планету від надмірного антропогенного втручання, бо він став продовженням людини. Тож порух миші тепер замінює безліч фізичних операцій, одним з неминучих наслідків яких є вплив на довкілля, і вплив, найчастіше, негативний.

Глобальна мережа перетворила планету і людство на щось дуже мале, обмежене у часі і просторі. Вони тепер вмістяться у долонях дитини, тож потребують захисту і турботи людини розумної.

Проблема, що виникла в Австралії, Антарктиді чи Африці, більше не сприймається як щось безмірно далеке, неспроможне бодай якось вплинути на буденне життя “тут і тепер”, у цивілізованіших регіонах планети.

Глобальні мережі поширили ареал “тут і тепер” на цілу планету. Тож коли щось негаразд на дні океану, на гірській вершині, в тропосфері, на полярній шапці чи в заштатному містечку N-ську – мережеве суспільство дійсно тим переймається, дійсно займається вирішенням проблеми, дійсно непокоїться за своє майбутнє і майбутнє планети. Адже це майбутнє тепер так просто побачити – інформаційні технології і мережеве знання миттєво збудують модель того, що з нами буде, коли ми не оберігатимем себе і планету.

Інформаційні технології сьогодні є екологічнішими за більшість інших видів активної людської діяльності, проте їх ще не можна назвати справді екологічними. Скажімо, ефективність інформаційних мереж напряму залежить від кількості користувачів, тобто, від кількості комп’ютерів, включених до мережі. Але, як зазначає член Європарламенту від Зеленої партії Керолайн Лукас, для виготовлення одного звичайного персонального комп’ютера потрібно від 15 до 19 тонн матеріалів. Це порівнювано з 25 тонами, потрібними для виготовлення автомобіля.

На кожен функціонуючий комп’ютер (використовуваний в середньому протягом 4 років) припадає 1,5 комп’ютери вироблених. А близько третини комп’ютерів ніколи не буває продано взагалі – через швидкість, з якою вони втрачають технологічну актуальність. Це означає, що затрачувані ресурси справді наближаються до рівня автомобіля.

Є потреба нової концепції розвитку інформаційних технологій – основаної на екоефективності, включаючи спільне використання машин, повторне застосування та ремонт.

Але це не єдиний шлях підвищення екологічної ефективності інформаційно-телекомунікаційних технологій. Нещодавно крупний виробник мобільних телефонів – компанія “Nokia” – повідомила про наміри протягом кількох років розробити мобільні телефони з біорозкладуваними компонентами.

В компанії вже розпочато випробування біорозкладуваних корпусів для мобільних телефонів, але поки що серед полімерних матеріалів не вдалося знайти таких, що були б при цьому стійкими до дії гострих предметів (тобто, матеріалів, на яких не залишається подряпин). Дослідники "Nokia" не сумніваються, що з часом рішення буде знайдено, однак це потребуватиме не менше двох років роботи.

Проблема утилізації використаних мобільних телефонів (як, проте, і комп’ютерів, периферійного обладнання, пейджерів тощо) стає гострішою з кожним роком. Обсяги виробництва продуктів інформаційно-телекомунікаційних технологій та частота їх заміни на нові моделі примушують компанії замислюватись над проблемою біодеградації.

Переходячи до другого аспекту проблеми – реалізації цілей екології через інформаційно-телекомунікаційні технології, ми знов зауважимо, що передумовою перетворення цих технологій на дієвий інструмент екології є масове їх поширення. Вони мають змінити спосіб життя достатньої кількості людей, родин, підприємств для того, аби ці зміни відбилися на суспільстві в цілому.

Тож основною перевагою такого значущого на сьогодні фактору людської діяльності як інформаційні мережі є не стільки їх “інформаційність”, скільки “електронність”, себто доступність, простота, зручність та швидкість

задоволення потреб користувача. За інакших обставин поява інформаційних мереж практично не позначилася б на способі життя людей, бо не здобула б їхньої прихильності.

Це означає, що тільки широке розповсюдження інформаційно-телекомунікаційних технологій забезпечить досягнення помітного екологічного ефекту. Екологія – явище, можливе лише в масштабах планети та людства.

Україна за цих обставин так само стає країною, чия екологія “виходить у гіперпростір”. Незважаючи на ускладнення економічного, політичного та правового характеру, український сегмент Мережі чинить значний вплив на екологічну ситуацію та на екоактивність громадян.

ВИСНОВКИ

На основі аналізу основних підходів до проектування архітектури ПС з врахуванням показників якості, показано що в більшості з них використовують не уніфіковані і не стандартизовані показники якості. Причому в основному це показники якості ПС, а не архітектури.

Залежність між цими групами показників досліджується методами логічного аналізу, а не формальними методами. Це приводить до суб'єктивних факторів при неоднозначності тлумачень показників, нечіткості, підміні понять.

Використання стандартизованих моделей якості ПС дозволяє використовувати при проектуванні архітектури ПС уніфіковану систему показників якості, уникнути двозначностей при трактуванні характеристик якості, розробити способи комунікації характеристик якості архітектури програмної системи з характеристиками якості готового програмного продукту.

Для виконання комунікації вимог, при складанні матриці парних порівнянь та при обчислення інтегральних характеристики якості одним з кроків при вирішення цих задач є встановлення пріоритетів характеристик якості. Цей параметр є досить суб'єктивним і алгоритм простого вибору

Не потрібно розв'язувати системи рівнянь та шукати власні числа матриць парних порівнянь для визначення пріоритетів характеристик.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Андон Ф. Основы инженерии качества программных систем / [Ф. Андон, Г. Коваль., Т. Коротун, Е. Лаврищева, В. Суслов] –К.: Академпериодика. – 2007. –672 с.
2. ISO/IEC 12207. Information technology – Software life cycle processes. 2008.
3. ISO/IEC 25010 Software engineering. Software product Quality, Reguirement and Evaluation (SquaRe), Quality Model, 2008.
4. Харченко О. Розробка та керування вимогами до програмного забезпечення на основі моделі якості / О. Харченко, В. Яцишин – Вісник ТДТУ – 2009. Том 14. №1. – с. 201-207
5. Яцишин В. Технологія оцінювання якості web-застосувань /В. Яцишин // Вісник ТДТУ. – 2009 – Том 14. – №4. – с. 132-140.
6. Харченко О.Г. Інструментальний засіб розробки та комунікації вимог якості до програмних систем / Харченко О.Г., Яцишин В.В., Райчев І.Е. // Науковий журнал «Інженерія програмного забезпечення» №2 – НАУ, Київ – 2010. – с. 29–34.
7. Пелещишин А.М. Методи побудови ефективних WWW-систем / А.М. Пелещишин // Вісник національного університету «Львівська політехніка»: Інформаційні системи та мережі. - №464. – 2002. –с. 240-255.
8. Саати Т. Принятие решений. Метод анализа иерархий / Т. Саати – М.: Радио и связь – 1993. – 315 с.
9. Черноруцкий И. Методы принятия решений / Черноруцкий И. – БХВ-Петербург – 2005. – 408 с.
10. Garlan D., Show M. Software Architecture. Perspectives on an Emerging Discipline, Englewood Clitts, NJ: Prentice Hall, 1996.
11. 6. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Д. Приёмы объектно-ориентированного проектирования. Патерны проектирования. СПб: Питер, 2010. -366с.

12. Glenn E., Krasner and Stephen T.Pope. F cookbook for using the model-view controller user interface paradigm in Smalltalk - 80. Jornal of Object-Orient Programming 1 (3):26-49, August 1998.

13. Фаулер М. Архитектура корпоративных программных приложений: Пер. с англ. – М.: Издательский дом «Вильямс», 2004 - 544с.

14. Басс Л., Клементс П., Кауман Р. Архитектура программного обеспечения на практике. 2-е издание. – СПб.: Питер, 2006. - 575с.

15. Nielsen J.: 2000, Designing Web Usability: The Practice of Simplicity, New Riders Publishing.

16. Коваль Г.І. Моделювання вимог до якості програмних систем оброблення даних/Г.І. Коваль, Г.Б. Мороз//Проблеми програмування, 2006. – №2,№3-с.273-244.

17. Харченко О.Г. Розроблення та керування вимогами до програмного забезпечення на основі моделі якості/О.Г. Харченко, В.В. Яцишин // Вісник ТДТУ.-Том 14.№1.-2009.-с.201-207.

18. ISO/IEC 9126. (1-4) Software Engineering-Product Quality / 2001-2004

19. Харченко О. Г. Проектування архітектури web-застосувань на основі моделі якості / О. Г. Харченко, І. О. Галай, І. О. Боднарчук, В. В. Яцишин // Інженерія програмного забезпечення. – 2010. – № 4. – С. 26-34.

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

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

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

ДОДАТКИ

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

М А Т Е Р І А Л И

VII НАУКОВО-ТЕХНІЧНОЇ КОНФЕРЕНЦІЇ

«ІНФОРМАЦІЙНІ МОДЕЛІ, СИСТЕМИ ТА ТЕХНОЛОГІЇ »

11–12 грудня 2019 року

ТЕРНОПІЛЬ
2019

УДК 004.7
А. Тарапата, А. Гулик
Тернопільський національний технічний університет імені Івана Пулюя

ВИКОРИСТАННЯ МОДЕЛЕЙ ЯКОСТІ ДЛЯ РОЗРОБКИ ВИМОГ

UDC 004.7
A. Tarapata, A. Hulyk
(Ternopil Ivan Puluj National Technical University, Ukraine)

REQUIREMENTS ENGINEERING ON THE BASE OF QUALITY MODELS

Питання розробки вимог якості до інформаційних систем (ІС) на програмному рівні на основі моделей якості розглядались в ряді публікацій [1], [2]. Тут розглядаються моделі трьох категорій якості, а саме якості у використанні, зовнішньої та внутрішньої якості.

На основі моделі якості у використанні розробляються загальні вимоги якості користувача або замовника. Модель вимог якості у використанні ІС можна представити у вигляді [3]

$$V_{use} = \{H^u_i,\ A^u_{ik},\ C^u_{ik},\ M^u_{ik}\},\ N_u,\ K = 1, S_i\ , \qquad (1)$$

тут H_{ui} – характеристика якості у використанні;
A_{uik} – атрибут характеристики якості;
C_{uik} – обмеження на значення атрибута;
$\underline{M_{uik}}$ – метрика атрибута.

Характеристики і метрики підбираються із стандарту [4], а атрибути і обмеження із вимог користувача та аналізу предметної області.

Вимоги зовнішньої якості представляються у вигляді структури моделі зовнішньої якості і інтерпретуються як вимоги до ІС в цілому, в тому числі і до архітектури. Ці вимоги записуються у вигляді

$$V_{ex} = \{H^e_i,\ П^e_{ik},\ A^e_{ik},\ C^e_{ik},\ M^e_{ik}\},\quad N_e,\ K = 1, R_i \qquad (3.2)$$

тут H^e_i - характеристики;
$П^e_{ik}$ - підхарактеристики зовнішньої якості;
$A^e_{ik},\ C^e_{ik},\ M^e_{ik}$ - атрибути, обмеження та метрики відповідно.

Комунікація (трасування) вимог якості (1) на структуру вимог (2) виконується з використанням методу SQFD [3]. На основі отриманих вимог зовнішньої якості формулюються вимоги якості до системи.

.

Література

1. Glenn E., Krasner and Stephen T.Pope. A cookbook for using the model-view controller user interface paradigm in Smalltalk – 80. Jornal of Object-Orient Programming 1 (3): 26-49, August 1998.
2. Фаулер М. Архитектура корпоративных программных приложений: Пер. с англ. – М.: Издательский дом «Вильямс», 2004. – 544 с.
3. Харченко О. Розробка та керування вимогами до програмного забезпечення на основі моделі якості / О. Харченко, В. Яцишин – Вісник ТДТУ – 2009. Том 14. №1. – с. 201-207.
4. ISO/IEC 12207:2008. Systems and software engineering – Software life cycle processes. – 123 p.