Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя
Факультет комп'ютерно-інформаційних систем і програмної інженерії
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

магістр

на тему: «Метод і засіб управління ризиками у гнучких методологіях розробки програмного забезпечення на основі адаптивної моделі SEI»

Виконала: студентка VI курсу, групи СПм-61 спеціальності 121 «Інженерія програмного забезпечення»

	(підпис)	Новицька Х.О. (прізвище та ініціали)
Керівник	(підпис)	Пастух О.А. (прізвище та ініціали)
Нормоконтроль	(підпис)	Стоянов Ю.М. (прізвище та ініціали)
Завідувач кафедри	(підпис)	Петрик М.Р. (прізвище та ініціали)
Рецензент	(підпис)	(прізвище та ініціали)

Тернопіль
2025

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*
Кафедра *Програмної інженерії*

ЗАТВЕРДЖУЮ
Завідувач кафедри *Петрик М.Р.*

«*28*» *листопада* 2025 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня *магістр*
(назва освітнього ступеня)

за спеціальністю *121 «Інженерія програмного забезпечення»*
(шифр і назва спеціальності)

студенту *Новицькій Христині Остапівні*
(прізвище, ім'я, по-батькові)

1. Тема проекту (роботи) *Метод і засіб управління ризиками у гнучких методологіях розробки програмного забезпечення на основі адаптивної моделі SEI*

Керівник проекту (роботи) *Пастух Олег Анатолійович, д.т.н., проф.*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від «*27*» *листопада* 2025 року №*4/7-1045*

2. Термін подання студентом завершеної роботи

3. Вихідні дані до роботи *Класифікація ризиків програмного забезпечення, модель SEI, технології управління ризиками, гнучкі методології управління проєктами.*

4. Зміст роботи (перелік питань, які потрібно розробити)
Вступ. 1. Аналіз гнучких методологій розробки та підходів до ідентифікації ризиків програмного забезпечення. 2. Розробка методу та проєктування засобу управління ризиками на основі моделі SEI. 3. Тестування та експериментальне застосування методу і засобу управління ризиками програмного забезпечення 4. Охорона праці та безпека в надзвичайних ситуаціях. Висновки

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
1. Актуальність і мета дослідження. 2. Задачі дослідження, об'єкт і предмет, наукова новизна і практична цінність дослідження. 3. Аналіз методів управління ризиками. 4. Структура моделі SEI 4. Метод управління ризиками. 5. Вимоги до засобу управління ризиками 6. Архітектура засобу управління ризиками. 7.Результати застосування методу і засобу управління ризиками 8. Висновки

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Охорона праці та	*Осухівська Г.М., зав. каф. КС*		
безпека в надзвичайних			
ситуаціях	*Стручок В.С., ст. викладач каф. ОХ*		

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	*Огляд літературних джерел. Постановка завдання на кваліфікаційну роботу.*	*28.11.2025р. – 03.12.2025р.*	*виконано*
2.	*Розробка методу та проєктування засобу управління ризиками на основі моделі SEI*	*03.12.2025р. – 06.12.2025р.*	*виконано*
3.	*Тестування та експериментальне застосування методу і засобу управління ризиками програмного забезпечення*	*06.12.2025р. – 09.12.2025р.*	*виконано*
4.	*Охорона праці та безпека в надзвичайних ситуаціях*	*09.12.2025р. – 11.12.2025р.*	*виконано*
5.	*Оформлення пояснювальної записки і графічного матеріалу*	*12.12.2025р. – 13.12.2025р.*	*виконано*
6.	*Попередній захист кваліфікаційної роботи магістра*	*15.12.2025р.*	*виконано*
7.	*Захист кваліфікаційної роботи магістра*		

Студентка _____ *Новицька Х.О.*
(підпис) (прізвище та ініціали)

Керівник проекту (роботи) _____ *Пастух О.А.*
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Новицька Х.О. Метод і засіб управління ризиками у гнучких методологіях розробки програмного забезпечення на основі адаптивної моделі SEI. Кваліфікаційна робота освітнього ступеня «магістр». Тернопільський національний технічний університет імені Івана Пулюя, кафедра програмної інженерії, група СПм-61 спеціальність 121 «Інженерія програмного забезпечення». Тернопіль, 2025. Сторінок 84, таблиць 8, рисунків 44, додатків 3, бібліографічних посилань 35.

Метою даної роботи є розроблення технології управління ризиками програмного забезпечення в умовах застосування гнучких методологій шляхом адаптації моделі SEI до особливостей Agile-процесів.

Об’єктом дослідження є процеси розроблення програмного забезпечення та управління ризиками у гнучких методологіях, а предметом - моделі керування ризиками, методологія Agile та методи оптимізації процесів управління ризиками під час розроблення програмного забезпечення.

Методи дослідження включають аналіз і узагальнення, формалізацію, проєктування і програмування, експериментальні методи та вимірювання.

У роботі розглянуто процес проєктування та реалізації програмного засобу підтримки управління ризиками програмного забезпечення із застосуванням гнучких методологій. Проведено аналіз існуючих підходів до управління ризиками та обґрунтовано доцільність використання моделі SEI для ідентифікації та класифікації ризиків на етапах життєвого циклу ПЗ. Запропоновано архітектуру багаторівневої програмної системи та реалізовано функціонал автоматизованого аналізу й моніторингу ризиків у межах Agile-проєктів.

Ключові слова: метод, засіб, ризик, управління, модель SEI.

ABSTRACT

Khrystyna Novytska. Method and Tool for Risk Management in Agile Software Development Methodologies Based on an Adaptive SEI Model. Qualification work of the educational level "Master". Ternopil IvanPuluj National Technical University, Department of Software Engineering, Group SPm-61, Specialty 121 "Software Engineering". Ternopil, 2025. Pages 84, tables 8, figures 44, annexes 3, bibliographic references 35.

The aim of this work is to develop a technology for software risk management under the application of agile methodologies by adapting the SEI model to the specifics of Agile processes.

The object of the research is the processes of software development and risk management in agile methodologies, while the subject of the research comprises risk management models, the Agile methodology, and methods for optimizing risk management processes during software development.

Research methods include analysis and generalization, formalization, design and programming, as well as experimental methods and measurement.

The paper considers the process of designing and implementing a software tool to support software risk management using agile methodologies. An analysis of existing risk management approaches is carried out, and the feasibility of applying the SEI model for risk identification and classification at different stages of the software life cycle is substantiated. A multilayer architecture of a software system is proposed, and the functionality for automated risk analysis and monitoring within Agile projects is implemented.

Keywords: method, tool, risk, management, SEI model.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

БД	База даних
ПЗ	Програмне забезпечення
BE	Back End
CASE	Computer Aided Software Engineering
ER	Entity Relations
FE	Front End
UML	Unified Modeling Language

ВСТУП

Актуальність теми. Сучасний етап розвитку індустрії програмного забезпечення характеризується високою динамікою змін, зростанням вимог до швидкості розробки та підвищенням очікувань кінцевих користувачів щодо якості готового продукту. Поряд із цим стрімкого поширення набувають хмарні технології, які дозволяють передавати частину відповідальності за інфраструктуру зовнішнім сервісам, тим самим прискорюючи та здешевлюючи розробку.

Попри значний прогрес у появі нових засобів, технологій і методологій створення ПЗ, якість реалізації програмних проєктів часто залишається недостатньо високою. Основною причиною є недосконаліть існуючих підходів до організації процесів управління, а особливо – до роботи з ризиками, що супроводжують розробку впродовж усього життєвого циклу.

Практика сучасних ІТ-компаній свідчить, що одними з найефективніших підходів до розроблення ПЗ залишаються гнучкі методології, серед яких Agile Modeling, Agile Unified Process, Scrum, Dynamic Systems Development Method, Extreme Programming та інші. Їхня ключова ідея полягає в адаптивності процесу та орієнтації на людей і взаємодію між ними, тоді як процеси і засоби відіграють другорядну роль.

Головною перевагою таких методологій є досягнення високого рівня задоволеності замовників, оскільки постійна комунікація між клієнтом і командою розробників дозволяє своєчасно реагувати на зміну вимог та коригувати пріоритети реалізованих функцій. У результаті функціональні можливості з найвищою цінністю впроваджуються першочергово, а команда краще розуміє очікування користувачів.

Водночас існує низка недоліків застосування гнучких методів: слабка формалізація документації, недостатньо опрацьовані механізми планування ітерацій, обмеженість інструментів для системного управління ризиками. Саме відсутність структурованих підходів до ідентифікації, оцінювання та

прогнозування ризиків часто стає причиною порушення термінів, підвищення вартості та зниження якості кінцевого продукту.

Таким чином, однією з актуальних задач сучасної інженерії програмного забезпечення є поєднання переваг гнучких підходів із формальними моделями управління ризиками. Особливої уваги потребує модель SEI (Software Engineering Institute), яка пропонує структурований підхід до класифікації та оцінювання ризиків та може бути адаптована до середовища Agile.

Метою даної роботи є розроблення технології управління ризиками програмного забезпечення в умовах застосування гнучких методологій шляхом адаптації моделі SEI до особливостей Agile-процесів.

Для досягнення поставленої мети необхідно вирішити такі **завдання**:

– здійснити аналіз сучасних наукових публікацій та практичних підходів до створення ПЗ з метою визначення стану застосування класичних і гнучких методологій у процесі розроблення програмних компонентів комп'ютерних систем;

– визначити специфічні риси формування програмних продуктів на основі Agile-методологій та оцінити їх вплив на процес керування ризиками;

– обґрунтувати можливість і доцільність використання моделі SEI для підтримки процесів ідентифікації, класифікації та моніторингу ризиків у гнучких командах розробки;

– інтегрувати критерії якості у процеси Agile для покращення прогнозування та зниження впливу ризиків;

– розробити та реалізувати метод керування ризиками на основі адаптованої моделі SEI, придатний до застосування в Scrum-процесах;

– спроєктувати й реалізувати програмний засіб, що підтримує процеси управління ризиками згідно з побудованою технологією та забезпечує автоматизацію оцінювання й прогнозування ризиків.

Об'єкт дослідження: процеси розроблення програмного забезпечення та управління ризиками у гнучких методологіях.

Предмет дослідження: моделі керування ризиками, методологія Agile та методи оптимізації процесів управління ризиками під час розроблення програмного забезпечення.

Методи дослідження включають:

– аналіз і узагальнення – для дослідження існуючих моделей ризик-менеджменту та гнучких методологій розробки ПЗ;

– формалізацію – при адаптації моделі SEI, побудові критеріїв оцінювання та інтеграції їх у Agile-процеси;

– проєктування і програмування – під час створення архітектури та реалізації програмного засобу підтримки керування ризиками;

– експериментальні методи та вимірювання – для перевірки ефективності запропонованої технології та розробленої системи в умовах реальних Agile-процесів.

Наукова новизна одержаних результатів. Наукова новизна полягає в наступному:

– уперше запропоновано адаптовану формалізовану модель управління ризиками програмного забезпечення в Agile-проєктах на основі класифікації SEI, яка інтегрує ризики вимог, архітектурних рішень та процесів розробки в єдину інформаційну модель і дозволяє здійснювати їх системну ідентифікацію та оцінювання на ранніх етапах життєвого циклу програмного продукту;

– удосконалено метод оцінювання пріоритетності ризиків у гнучких методологіях розробки шляхом поєднання експертних оцінок, метрик якості вимог та результатів аналізу критичного шляху проєкту, що дало змогу підвищити обґрунтованість прийняття управлінських рішень щодо пом’якшення ризиків без порушення ітеративної природи Agile-процесів;

– набули подальшого розвитку підходи до автоматизації процесів управління ризиками програмного забезпечення за рахунок розроблення багаторівневої архітектури програмного засобу, яка забезпечує трасування ризиків, вимог і метрик якості між ітераціями Agile-проєкту, що сприяє підвищенню

прозорості процесу розробки та зниженню ймовірності критичних відхилень під час реалізації програмних проєктів.

Практична цінність результатів дослідження. Розроблений програмний засіб забезпечує автоматизацію процесів обліку, аналізу та моніторингу ризиків у межах ітерацій Agile-проєктів, що сприяє підвищенню прозорості управлінських рішень, зменшенню ймовірності критичних відхилень у термінах та бюджеті, а також покращенню якості програмного продукту.

Апробація. Окремі результати дослідження апробовано на X міжнародній науково - технічній конференції молодих учених і студентів «Актуальні задачі сучасних технологій» (24-25 листопада 2021 р.) Тернопільського національного технічного університету імені Івана Пулюя та на IX науково-технічній конференції Тернопільського національного технічного університету імені Івана Пулюя «Інформаційні моделі, системи та технології» (8-9 грудня 2021 року) у вигляді тез конференцій.

1. Пастух О.А., Новицька Х.О. Моделювання процесів управління ризиками в життєвому циклі програмного забезпечення із застосуванням UML та адаптованої SEI-моделі. Матеріали XIV міжнародної науково - технічної конференції молодих учених і студентів «Актуальні задачі сучасних технологій» (11-12 грудня 2025 р.) Тернопільського національного технічного університету імені Івана Пулюя. Тернопіль: ТНТУ. 2025. С. 315-318.

2. Пастух О.А., Новицька Х.О. Архітектура програмної системи управління ризиками на основі адаптованої моделі SEI. Матеріали XIII науково-технічної конференції Тернопільського національного технічного університету імені Івана Пулюя «Інформаційні моделі, системи та технології» (17-18 грудня 2025 року). Тернопіль: ТНТУ. 2025. С. 186.

Структура роботи. Робота складається з пояснювальної записки та графічної частини. Пояснювальна записка складається з вступу, 4 розділів, висновків, списку використаної літератури та додатків. Обсяг роботи: пояснювальна записка – 83 арк. формату А4, графічна частина – 35 слайди.

1 АНАЛІЗ ГНУЧКИХ МЕТОДОЛОГІЙ РОЗРОБКИ ТА ПІДХОДІВ ДО ІДЕНТИФІКАЦІЇ РИЗИКІВ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

У першому розділі проведено аналіз гнучких методологій розробки програмного забезпечення і встановлено, що вони дозволяють забезпечити високу адаптивність процесів, однак потребують формалізації процесу управління ризиками. Досліджено модель SEI як інструмент ідентифікації та класифікації ризиків програмних проєктів та можливість її інтеграції у процеси гнучких методологій розробки. Обґрунтовано доцільність використання UML як засобу структурованого опису програмного проєкту з метою підтримки процесу ідентифікації ризиків на ранніх етапах життєвого циклу.

1.1. Гнучкий підхід до організації розробки програмного забезпечення за методологією Scrum

У межах методології Scrum проєкт поділяється на три основні етапи, кожен із яких має власну мету та набір завдань.

Підготовчий етап охоплює планування загальної стратегії проєкту, визначення ключових вимог до майбутнього продукту, що формують так званий реєстр вимог продукту (Product Backlog). На цьому етапі створюється загальне бачення архітектури системи, окреслюються її основні компоненти та встановлюються пріоритети розробки. Основний етап (ігрова стадія) передбачає безпосередню реалізацію продукту шляхом циклічних ітерацій, які отримали назву спринтів (Sprint). Кожен спринт триває, як правило, від двох до чотирьох тижнів і розглядається як завершений цикл розробки, що містить елементи класичної каскадної моделі. Після завершення кожного спринту формується нова версія продукту, яка має бути повністю працездатною [1].

Кожен спринт починається із зустрічі з планування (Sprint Planning Meeting), під час якої команда визначає набір функцій, що увійдуть до списку завдань спринту (Sprint Backlog). Щодня відбуваються короткі робочі наради (щоденні

збори Scrum – Daily Scrum Meeting), де кожен учасник повідомляє про прогрес у виконанні завдань, труднощі та плани на поточний день. По завершенні ітерації проводиться підсумкова демонстраційна зустріч (Sprint Review Meeting), під час якої замовнику представляють результати реалізації. Завершальний етап (післяігрова стадія) спрямований на підготовку кінцевої версії продукту до випуску та впровадження. На цьому етапі здійснюється остаточне тестування, перевірка працездатності та оформлення документації [1-3].

Загальна логічна структура процесу Scrum подана на рис. 1.1, а ітераційна модель спринту – на рис. 1.2.

Рисунок 1.1 – Загальна структура процесу Scrum

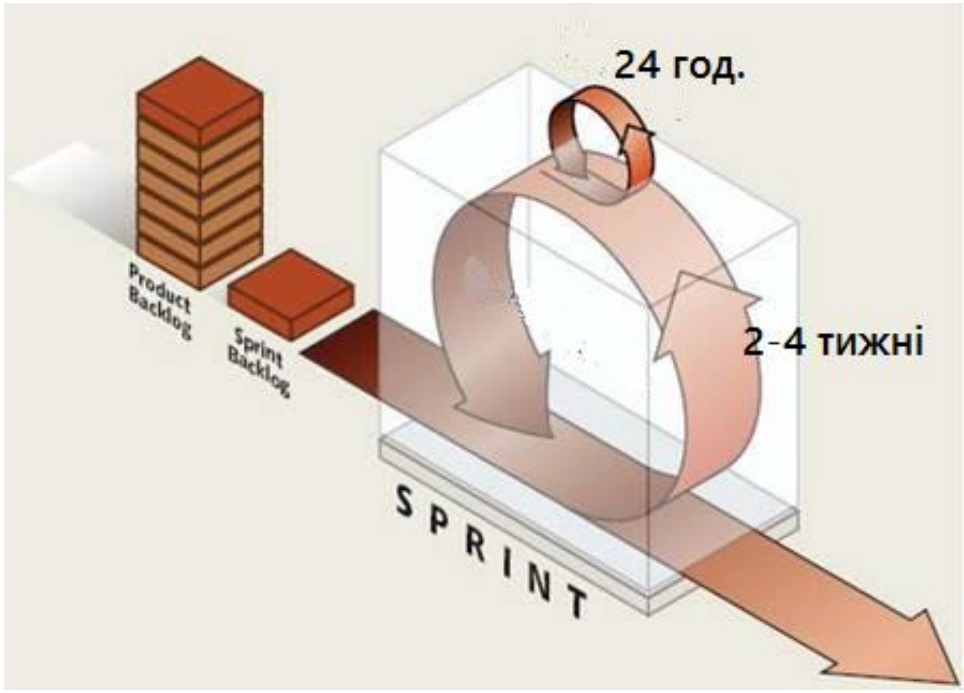

Рисунок 1.2 – Модель виконання спринту в методології Scrum

Під час ітераційного циклу відповідальність за реалізацію функціоналу покладається виключно на команду розробників. Scrum не передбачає жорсткого поділу на ролі, що забезпечує гнучкість у прийнятті рішень і відповідальність усіх членів колективу. Взаємодія з керівництвом та замовником здійснюється після завершення кожної ітерації. Переривання спринту допускається лише у виняткових випадках, якщо виникають критичні обставини, що впливають на хід проєкту.

1.2. Методика розроблення програмних систем на основі підходу динамічних систем

Метод розроблення динамічних систем (DSDM – Dynamic Systems Development Method**)** є гнучкою методологією керування проєктами, що зосереджена на швидкому отриманні результату при збереженні стабільності бюджету. Вперше метод було представлено у Великій Британії у 1994 році групою компаній, які об’єдналися в некомерційний консорціум DSDM Consortium. Метою створення цієї методики було вдосконалення практик швидкої розробки (RAD) та ітеративного підходу [4-6].

Особливістю DSDM є те, що вона створювалася спільнотою компаній, а не окремим автором, що забезпечило їй високий рівень зрілості та універсальності. Спочатку методологію застосовували переважно у сфері створення програмного забезпечення, проте з часом її принципи почали використовувати й в інших галузях.

Основними перевагами DSDM є простота, гнучкість, масштабованість і ефективність. Водночас її впровадження потребує певної підготовки організації, оскільки метод вимагає змін у структурі управління та культурі взаємодії між учасниками команди [5].

DSDM розглядається як одна з найрозвиненіших методологій гнучкої розробки, адже зосереджується не стільки на програмуванні, скільки на організації процесу створення програмного продукту. У цьому аспекті вона має спільні риси

зі Scrum, проте робить більший акцент на управлінських і комунікаційних процесах.

Початком роботи в межах DSDM є аналіз здійсненності проєкту та визначення його бізнес-контексту. На ранньому етапі проводяться короткі семінари, під час яких команда розробників ознайомлюється зі специфікою предметної області, формулюються ключові вимоги, архітектурні рішення та загальний план реалізації [6].

Філософія DSDM базується на низці базових принципів, дотримання яких є обов’язковим для успіху проєкту.

Перший принцип – це активна участь користувачів. Постійна взаємодія з кінцевими користувачами зменшує кількість помилкових рішень і сприяє створенню продукту, що відповідає реальним потребам бізнесу.

Наступний принцип стосується повноважень команди щодо прийняття рішень. Делегування відповідальності членам команди дозволяє уникнути бюрократичних затримок і підвищує ефективність роботи.

Третій принцип передбачає регулярне постачання результатів. Часте створення проміжних версій забезпечує швидке виявлення помилок і підвищує якість кінцевого продукту [7].

Четвертий принцип стосується фокусу на бізнес-цілях. Основною метою є задоволення актуальних потреб замовника, навіть якщо для цього потрібно тимчасово спростити технічні аспекти.

Дотримання принципу ітеративності та інкрементальності передбчає те, що розробка поділяється на серію ітерацій, кожна з яких додає нову функціональність до системи. Це дозволяє поступово досягати повної відповідності вимогам.

Наступний принцип –це можливість скасування дій. Будь-яке рішення чи зміна можуть бути переглянуті або відмінені без значних втрат завдяки невеликим крокам розробки. Сьомий принцип передбачає фіксацію високорівневих вимог. Основні вимоги до продукту визначаються на початку, що мінімізує ризики постійних змін під час реалізації [7].

Принцип дотримання інтегрованого тестування стосується перевірки якості проводиться паралельно з розробкою, починаючи з аналізу вимог і протягом усього життєвого циклу. Останнім принципом є співпраця всіх зацікавлених сторін. Успіх проєкту залежить від постійної комунікації між менеджерами, технічними фахівцями та користувачами. Загальна структура процесу DSDM охоплює три взаємопов'язані цикли, наведені на рис. 1.3.

1. Цикл моделювання функцій – створення аналітичних моделей, прототипів та опису функціональності системи.

2. Цикл проектування та реалізації – розроблення робочої версії продукту через кілька ітерацій із забезпеченням належної якості.

3. Цикл впровадження – підготовка продукту до експлуатації, навчання користувачів, оформлення супровідної документації.

Рисунок 1.3 – Етапи процесу розроблення програмного продукту за методологією DSDM

Методологія DSDM визначає одинадцять основних ролей у команді, кожна з яких має власну зону відповідальності. Такий розподіл забезпечує чіткість у виконанні завдань, синхронізацію між учасниками та ефективне досягнення цілей проєкту.

1.3. Ідентифікація та класифікація ризиків відповідно до моделі SEI

З метою формалізації та спрощення процесу ідентифікації ризиків у програмних проєктах доцільно використовувати класифікаційні підходи, що ґрунтуються на перевірених інженерних практиках.

Однією з найбільш поширених і практично орієнтованих є класифікація ризиків, запропонована Software Engineering Institute (SEI). Вона сформована з урахуванням типових процесів життєвого циклу програмного забезпечення та охоплює ключові групи ризиків, пов’язані з характеристиками програмного продукту, середовищем розробки, організацією процесів і проєктними обмеженнями [9].

Оскільки у даній роботі основна увага зосереджена на аналізі ризиків на ранніх етапах життєвого циклу, пріоритетним є дослідження внутрішніх технічних ризиків, що виникають під час формування вимог, проєктування та початкової реалізації програмних складових. Відповідна класифікація технічних аспектів ризиків за моделлю SEI наведена в табл. 1.1.

Таблиця 1.1 – Класифікація технічних ризиків програмного проєкту

Клас джерела ризику	**Загальна характеристика класу**	**Компонент ризику**	**Описовий атрибут**
Технічні ризики розробки ПЗ	Ризики, що виникають у процесі створення програмного забезпечення на різних етапах життєвого циклу та пов’язані з властивостями продукту	Вимоги	Стійкість до змін
			Повнота опису
			Однозначність формулювань
			Коректність відображення потреб
			Реалістичність реалізації
			Рівень інноваційності
			Масштаб проєкту

Продовження таблиці 1.1

Клас джерела ризику	**Загальна характеристика класу**	**Компонент ризику**	**Описовий атрибут**
Технічні ризики розробки ПЗ	Ризики, що виникають у процесі створення програмного забезпечення на різних етапах життєвого циклу та пов’язані з властивостями продукту	Архітектура та проєктні рішення	Функціональні можливості
			Рівень складності
			Узгодженість інтерфейсів
			Продуктивність системи
			Можливість тестування
			Апаратні обмеження
			Залежність від стороннього ПЗ
		Характеристики якості	Зручність супроводу
			Надійність
			Захищеність даних
			Безпека
			Вплив людського чинника
			Якість специфікацій

Згідно з наведеною класифікацією, технічні ризики охоплюють ризики, пов’язані з вимогами до програмного забезпечення, архітектурними рішеннями, складністю проєкту, нефункціональними характеристиками, а також впливом зовнішніх обмежень. Кожен із цих елементів деталізується через набір атрибутів, які можуть бути об’єктом аналізу на відповідних стадіях життєвого циклу [6-7].

Для збору вихідної інформації щодо потенційних ризиків на практиці застосовуються різні експертні методи, серед яких найбільш поширеними є:

– експертні опитування;
– метод мозкового штурму;
– метод Делфі;
– метод карток Кроуфорда.

Усі перелічені підходи базуються на залученні кваліфікованих фахівців, які на основі власного досвіду, професійних знань та наданої інформації здійснюють попередню оцінку стану програмного проєкту. Якість результатів ідентифікації ризиків у значній мірі залежить від повноти, структурованості та точності опису програмної системи [7].

Саме детальний і формалізований опис проєкту є ключовим чинником, що впливає на достовірність експертних висновків. Хоча задача формування такого опису є достатньо складною, сучасна практика програмної інженерії пропонує низку методик, які дозволяють досягти необхідного рівня формалізації [9].

Насамперед ідеться про методи, що застосовуються на етапах аналізу та проєктування програмного забезпечення, зокрема:

– методи структурного системного аналізу (ER-діаграми, SADT, IDEF);
– мову об’єктно-орієнтованого моделювання UML.

На сьогодні UML фактично є загальновизнаним стандартом опису програмних систем, що розробляються з використанням об’єктно-орієнтованих технологій. З огляду на це, використання UML для формування вхідної інформації в процесі ідентифікації ризиків є найбільш доцільним.

Модель складної програмної системи, побудована з використанням UML, охоплює чотири взаємопов’язані подання:

– логічне подання;
– подання реалізації;
– подання процесів виконання;
– подання розміщення.

Кожне з наведених подань включає відповідний набір UML-діаграм, які відображають окремі аспекти майбутнього програмного продукту. Вибір конкретних діаграм визначається інформаційними потребами експертів з аналізу ризиків. Узагальнену відповідність типів UML-діаграм та аспектів опису проєкту наведено в табл. 1.2.

Таблиця 1.2 – Використання UML-діаграм для опису програмного проєкту

Тип UML-діаграми	**Інформаційний аспект проєкту**
Діаграма варіантів використання (Use Case)	Формалізація функціональних вимог і ролей користувачів
Діаграма класів	Попереднє уявлення про структуру та архітектуру програмної системи
Діаграми поведінки та взаємодії	Опис сценаріїв виконання функцій і обміну повідомленнями
Діаграма розміщення	Відображення апаратної конфігурації та середовища виконання
Діаграма компонентів	Перелік програмних модулів і зовнішніх інструментів, що використовуються у системі

Використання UML дозволяє подати технічну інформацію у чітко структурованому та формалізованому вигляді, що значно полегшує її аналіз. Водночас слід зауважити, що такий підхід орієнтований передусім на технічні аспекти розробки, унаслідок чого не всі групи ризиків можуть бути ідентифіковані виключно на основі UML-моделей.

Відповідно до класифікації SEI (табл. 1.1), за допомогою UML можливо повністю або частково оцінити ризики, пов’язані з вимогами та характеристиками

проєкту. Перелік елементів класів ризиків, що піддаються ідентифікації з використанням UML, наведено в табл. 1.3.

Таблиця 1.3 – Елементи класів ризиків, що ідентифікуються на основі UML

Клас джерела ризику	**Компонент**	**Атрибут для аналізу**
Технічні аспекти розробки	Вимоги	Стабільність вимог
		Повнота функціонального опису
		Відсутність неоднозначностей
		Достовірність даних
		Реалізованість у заданих умовах
		Масштабованість
	Архітектура	Функціональність
		Структурна складність
		Взаємодія інтерфейсів
		Швидкодія
		Тестованість
		Обмеження апаратних ресурсів
		Використання зовнішніх компонентів
	Нефункціональні характеристики	Надійність
		Захист інформації
		Безпека
		Якість технічної документації

Разом з тим існує низка атрибутів, для аналізу яких UML не забезпечує достатнього рівня інформації. До них належать новизна рішень, зручність супроводу та вплив людського фактору. Відповідні елементи наведено в табл. 1.4.

Таблиця 1.4 – Елементи ризиків, що не ідентифікуються засобами UML

Клас джерела ризику	**Загальна характеристика**	**Компонент**	**Атрибут**
Технічні ризики розробки ПЗ	Ризики, пов’язані з особливостями організації процесів та людським фактором	Вимоги	Новизна предметної області
		Нефункціональні характеристики	Зручність подальшого супроводу
			Вплив людського чинника

З урахуванням наведеного можна стверджувати, що використання UML дозволяє експертам оцінити більшість (17 із 20) атрибутів ризиків, визначених у моделі SEI. Це істотно спрощує організацію процесу аналізу ризиків та сприяє підвищенню якості управлінських рішень на ранніх стадіях життєвого циклу програмного забезпечення [10].

Аналогічний підхід може бути використаний і для збереження інформації про раніше реалізовані програмні проєкти. Наявність архівних UML-моделей забезпечує швидкий доступ до структурованих даних, що значно підвищує ефективність повторного аналізу ризиків [11].

Хоча UML не є єдиним засобом моделювання програмних систем, у разі застосування об’єктно-орієнтованих методологій його використання надає найбільші переваги. Водночас серед обмежень UML слід виділити неможливість прямого представлення таких чинників, як часові та вартісні обмеження, а також

окремі організаційні аспекти, для аналізу яких використовуються спеціалізовані інструменти управління проєктами.

Незважаючи на зазначені обмеження, мова моделювання UML залишається ефективним і практично доцільним інструментом підтримки процесу ідентифікації ризиків, особливо під час розробки складних програмних та автоматизованих систем керування.

1.4. Метод ідентифікації ризиків на основі моделі SEI

Відповідно до підходу, запропонованого у моделі SEI (Software Engineering Institute), існує вісімнадцять основних типів джерел ризиків, що найчастіше зустрічаються у процесі реалізації програмних проєктів [12]. Сукупність цих джерел можна формально подати у вигляді множини:

$$RS = \{rs_1, \ldots, rs_{18}\}, \tag{1.1}$$

де rs_i – окреме джерело потенційного ризику, $i = 1, \ldots, 18$.

Одним із ключових класів є технічні ризики, які охоплюють аспекти функціонування, якості та підтримуваності програмного забезпечення. Їх множину можна визначити так:

$$TRS = \{rs_1, \ldots, rs_7\}, \tag{1.2}$$

де $TRS \subset RS$ – підмножина технічних ризиків, до якої належать:

- rs_1 – характеристики функціональності програмного продукту;
- rs_2 – показники якості реалізації;
- rs_3 – параметри надійності;
- rs_4 – придатність до практичного застосування;
- rs_5 – показники часової ефективності (продуктивності);
- rs_6 – рівень зручності супроводу;

– rs_7 – можливість повторного використання програмних компонентів.

Окрему групу становлять ризики, пов’язані з фінансуванням та вартісними обмеженнями, які можна описати підмножиною:

$$CRS = \{rs_8,...,rs_{10}\}, \quad (1.3)$$

де $CRS \subset RS$ – підмножина вартісних ризиків, що включає:

– rs_8 – обмеження загального бюджету проєкту;

– rs_9 – недостатній рівень фінансування;

– rs_{10} – нереалістичність оцінки витрат і кошторису.

Вагомий вплив на хід реалізації програмного продукту мають також ризики планування, які характеризують рівень організованості процесу та ефективність управління строками. Формально їх можна подати так:

$$PRS = \{rs_{11}, \dots, rs_{13}\}, \quad (1.4)$$

де $PRS \subset RS$ – підмножина планових ризиків, до складу якої входять:

– rs_{11} – гнучкість у коригуванні планів;

– rs_{12} – ймовірність порушення встановлених термінів етапів життєвого циклу;

– rs_{13} – недостатній рівень реалізму під час планування етапів розробки.

Ще один важливий клас становлять ризики управління та супровідних процесів, що супроводжують життєвий цикл проєкту. Вони описуються множиною:

$$MRS = \{rs_{14}, \dots, rs_{18}\}, \quad (1.5)$$

де $MRS \subset RS$ – підмножина ризиків, пов’язаних з процесами керування та допоміжними процедурами, до якої належать:

– rs_{14} – стратегічні аспекти управління проєктом;

- rs_{15} – якість планування робіт;
- rs_{16} – адекватність методів оцінювання результатів;
- rs_{17} – повнота та достовірність документування;
- rs_{18} – точність прогнозування розвитку проєкту.

Таким чином, проведена класифікація дозволяє системно охопити всі потенційні джерела ризиків, що можуть впливати на ефективність, вартість і терміни реалізації програмного продукту.

1.5. Висновки до першого розділу

1. На основі аналізу гнучких методологій розробки програмного забезпечення встановлено, що Scrum і DSDM забезпечують високу адаптивність процесів, проте потребують доповнення формалізованими механізмами управління ризиками. Це обґрунтовує доцільність інтеграції моделей ризик-менеджменту в Agile-процеси з метою підвищення передбачуваності результатів розробки.

3. Досліджено модель SEI як інструмент формалізованої ідентифікації та класифікації ризиків програмних проєктів. Визначено основні групи джерел ризиків (технічні, вартісні, планові та управлінські), що дозволяє комплексно охопити фактори, які впливають на якість, строки та вартість реалізації програмного продукту.

4. Обґрунтовано доцільність використання UML як засобу структурованого опису програмного проєкту з метою підтримки процесу ідентифікації ризиків на ранніх етапах життєвого циклу. Встановлено, що застосування UML-діаграм дає змогу ідентифікувати більшість атрибутів технічних ризиків відповідно до моделі SEI.

5. За результатами аналізу визначено, що поєднання Agile-методологій, моделі SEI та UML-моделювання створює методологічну основу для розробки адаптованої технології управління, оцінювання та прогнозування ризиків програмного забезпечення, що й визначає напрям подальших досліджень у наступних розділах роботи.

2 РОЗРОБКА МЕТОДУ ТА ПРОЄКТУВАННЯ ЗАСОБУ УПРАВЛІННЯ РИЗИКАМИ НА ОСНОВІ МОДЕЛІ SEI

У другому розділі запропоновано метод управління ризиками на основі моделі SEI та моделей стандарту ISO/IEC 25010. Даний метод адаптований до використання у гнучких (Agile) методологіях розробки і дає можливість пов’язати ризики з вимогами до якості програмного продукту та забезпечити їх системне врахування на всіх етапах життєвого циклу ПЗ. Окрім цього, у розділі запропоновано багаторівневу архітектуру програмного засобу управління ризиками. Модель архітектури включає рівні представлення, прикладної логіки, доменної моделі, доступу до даних і бази даних. Такий підхід забезпечує розділення відповідальностей, слабке зв’язування компонентів та можливість подальшого масштабування або інтеграції системи з іншими програмними рішеннями.

2.1. Метод управління ризиками на основі моделі SEI та моделей стандарту ISO/IEC 25010

Сучасна інженерія програмного забезпечення характеризується використанням широкого спектра моделей життєвого циклу. Вибір конкретної моделі визначається типом програмного продукту, зокрема його призначенням, рівнем критичності та вимогами до надійності (наприклад, програмне забезпечення масового використання, медичні або реального часу системи) [13].

Моделі життєвого циклу програмного забезпечення, окрім базових етапів, включають набір процесів і підпроцесів, орієнтованих на забезпечення необхідного рівня якості кінцевого продукту з урахуванням часових і бюджетних обмежень проєкту.

Одним із ключових аспектів програмної інженерії є ідентифікація, аналіз та управління ризиками на різних стадіях життєвого циклу. Результати такого управління суттєво залежать від того, які методи та інструменти застосовуються на

кожному етапі. У даній роботі запропоновано метод управління та моніторингу ризиків у процесі розробки програмного забезпечення з використанням гнучких методологій. В основі методу лежить підхід комунікації вимог до якості з урахуванням моделі ризиків SEI [14].

Під ризиком програмного проєкту розуміється ймовірність зниження якості кінцевого продукту, перевищення бюджету, порушення термінів виконання або навіть зриву реалізації проєкту. Такі ризики, як правило, пов’язані з недосконалістю методів, технологій та процесів, що застосовуються на різних стадіях життєвого циклу.

З позиції керування проєктом особливе значення мають технічні ризики. Саме тому критично важливо ідентифікувати їх на рівні вимог, оскільки вимоги формують основу майбутньої еволюції програмного продукту.

У загальному випадку всі ризики поділяються на внутрішні та зовнішні. Внутрішні ризики пов’язані з особливостями самого процесу розробки та можуть бути контрольовані командою за допомогою відповідних інструментів і методів. Зовнішні ризики зумовлені факторами, що не залежать від розробників (ринкові умови, законодавчі обмеження тощо).

Інститут управління проєктами (PMI) запропонував системний підхід до управління ризиками, який викладено у PMBOK та включає такі процеси [15]:

– планування управління ризиками;
– ідентифікацію ризиків;
– якісний аналіз ризиків;
– кількісний аналіз ризиків;
– планування реакцій на ризики;
– моніторинг і контроль ризиків.

Серед практичних підходів до управління ризиками особливу роль відіграє модель, розроблена Software Engineering Institute (SEI).

Для аналізу та оцінювання ризиків програмного забезпечення застосовуються як формалізовані, так і неформальні методи. Вибір конкретного

підходу залежить від призначення програмного продукту та критичності можливих наслідків відмов [15].

До найбільш відомих формалізованих методів належать:

– аналіз дерева подій (ETA);
– аналіз дерева відмов (FTA);
– аналіз видів і наслідків відмов (FMEA).

Ці методи є ефективними на етапах аналізу вимог, проєктування архітектури та реалізації складних програмних систем, оскільки дозволяють виявляти потенційні сценарії відмов і оцінювати їх наслідки.

Метод ETA дозволяє простежити ланцюжок можливих наслідків відмови компонента, однак не дає інформації про першопричини таких відмов.

Методи FMEA, навпаки, орієнтовані на виявлення можливих типів відмов та умов їх виникнення, що робить їх ефективними для аналізу причинно-наслідкових зв’язків.

Класифікація ризиків, запропонована SEI, є зручною основою для виявлення типових джерел ризиків програмних проєктів, проте потребує адаптації до конкретних умов.

SEI визначає три основні підходи до управління ризиками:

1. Software Risk Evaluation (SRE) – формалізований метод ідентифікації та аналізу ризиків на ранніх стадіях;
2. Continuous Risk Management (CRM) – безперервне управління ризиками протягом життєвого циклу;
3. Team Risk Management (TRM) – управління ризиками з урахуванням командної взаємодії та участі стейкхолдерів.

Загальний процес аналізу ризиків включає формування експертної групи, підготовку питань, вибір методів аналізу, ранжування ризиків, розробку механізмів реагування та підготовку звітної документації.

Як зазначалось у розділі 1, модель SEI описує 18 типових джерел ризиків, які можуть бути представлені у вигляді множин і підмножин технічних, вартісних,

планових та організаційних ризиків. У роботі запропоновано представити дану модуль у вигляді ієрархічної структури (рис. 2.1).

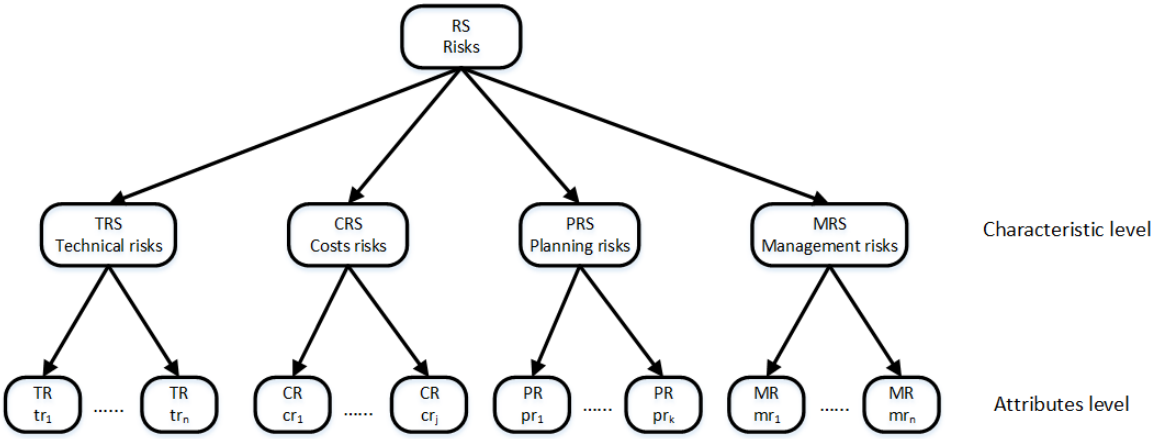

Рисунок 2.1 – Ієрархічна структура моделі ризиків SEI

Для кожної групи ризиків визначається набір атрибутів, що дозволяє виконувати подальшу кількісну оцінку та інтеграцію ризиків у процес розробки.

З іншої сторони, відомо, що модель якості програмного забезпечення визначається стандартом ISO/IEC 25010 та включає моделі якості у використанні, зовнішньої та внутрішньої якості (рис. 2.2).

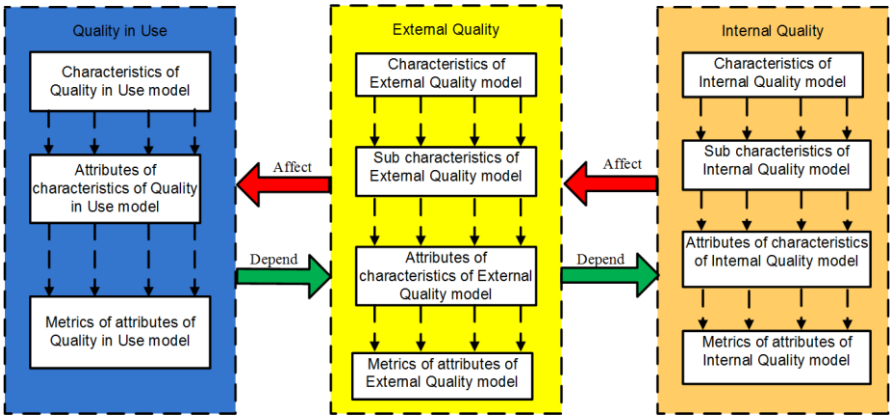

Рисунок 2.2. – Структура моделей якості ISO/IEC 25010

Інтеграція ризиків SEI з вимогами до якості дозволяє сформувати ієрархічну структуру «вимога–ризик» (рис. 2.3).

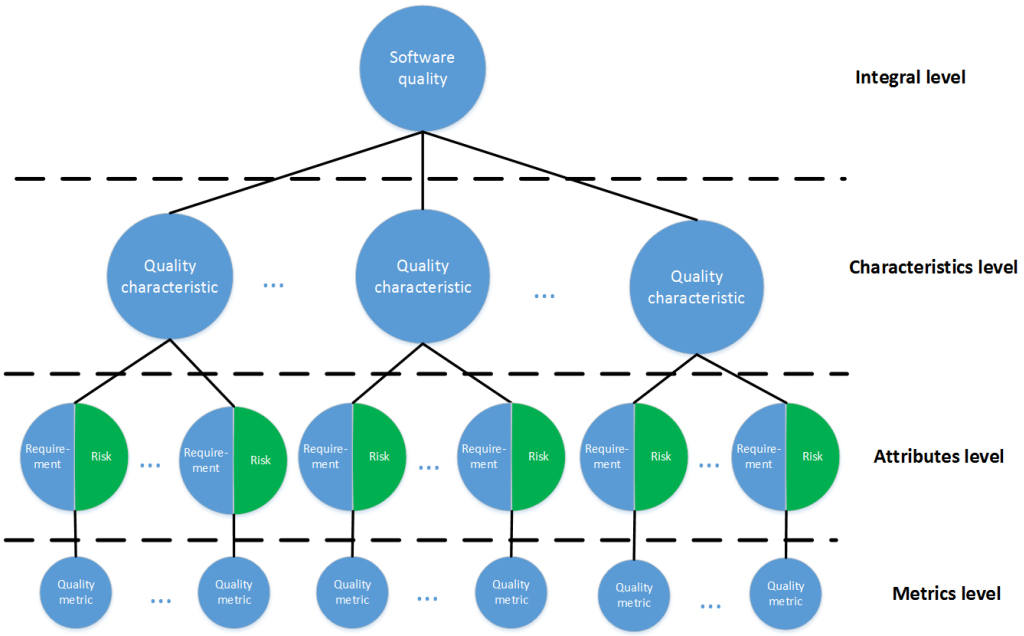

Рисунок 2.3 – Модель управління ризиками на основі адаптованої моделі SEI та моделі якості ISO/IEC 25010

Запропонований метод передбачає інтеграцію ризиків у всі ключові етапи Agile-процесу: від аналізу потреб замовника до планування спринтів і виконання завдань.

Ризики пов’язуються з вимогами, а їхні пріоритети враховуються під час формування архітектури, декомпозиції завдань і планування спринтів, що дозволяє здійснювати системне управління ризиками протягом усього життєвого циклу. Це забезпечує підвищення повноти, трасовності та керованості ризиків на ранніх стадіях життєвого циклу програмного забезпечення та створює основу для подальшої автоматизації процесів управління ризиками.

2.2. Аналіз домену та визначення вимог до засобу управління ризиками у гнучких методологіях розробки ПЗ

Комплексне вивчення предметної області та ключових процесів життєвого циклу програмного забезпечення, зокрема тих, що пов’язані з управлінням ризиками, є важливою передумовою для побудови програмної системи, спрямованої на підвищення результативності розробки програмних продуктів.

Аналіз зазначених процесів дозволяє визначити основні вимоги до системи, виокремити критичні аспекти та сформувати концептуальну модель, що стане основою для подальшого проектування.

У процесі дослідження предметної області можуть застосовуватися різноманітні методи: структурний аналіз, об’єктно-орієнтоване чи візуальне моделювання, а також інші підходи, які забезпечують формалізацію знань про систему. Серед них найбільш ефективним у контексті проектування складних програмних систем є об’єктно-орієнтований підхід, що дозволяє логічно впорядковувати інформацію й відтворювати реальні системи у вигляді взаємопов’язаних об’єктів [16].

Одним із найзручніших та загальноприйнятих засобів об’єктно-орієнтованого моделювання є UML. Дана мова забезпечує широкий спектр графічних нотацій для опису структури та поведінки системи. У межах аналізу предметної області особливу цінність становлять такі діаграми, як діаграма варіантів використання, діаграма класів, діаграма компонентів та діаграма видів діяльності.

Оскільки в даному випадку необхідно відобразити функціональні можливості майбутньої системи з урахуванням її практичного застосування, доцільно використати діаграму прецедентів. Вона дозволяє представити взаємодію користувачів із системою та визначити основні сценарії їхньої роботи [17].

Реалізація підходу управління ризиками в процесі проектування програмних рішень, який інтегрується у методології Agile та спирається на адаптовану модель SEI, потребує участі кількох ролей: системний аналітик, експерт з ризиків та менеджер проєкту. На рис. 2.4 відображено діаграму прецедентів, що демонструє основні функції експерта в межах Agile-процесів.

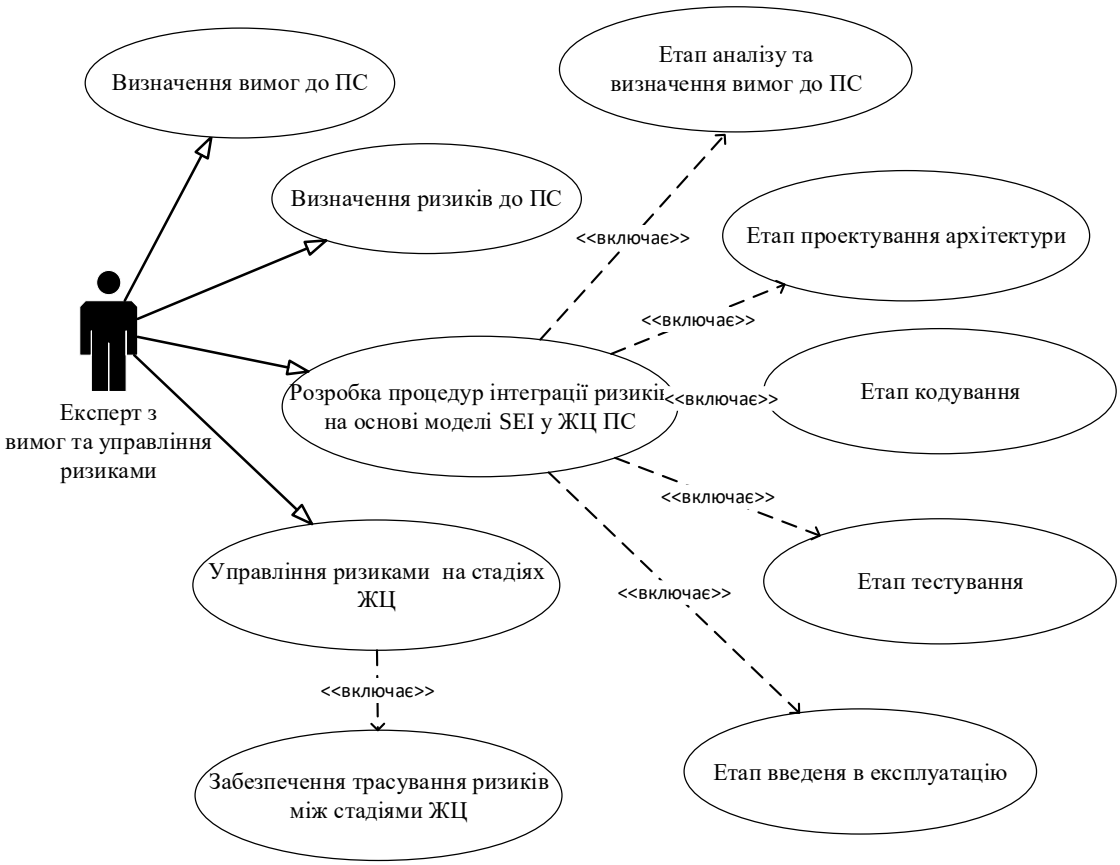

Рисунок 2.3 – Діаграма прецедентів «Експерт»

Експерт з ризиків визначає вимоги до якості програмного продукту, ідентифікує ризики, проводить їх аналіз та забезпечує контроль у межах життєвого циклу. Сфера відповідальності експерта з ризиків включає виявлення потенційних загроз для програмного продукту, формування відповідних вимог та забезпечення їх інтеграції у процеси планування й виконання робіт. Експерт також координує процедури оцінювання та підтримує актуальність інформації про ризики.

Менеджер проєкту здійснює планування, організацію, відстеження та коригування процесів розробки. Менеджер проєкту, за умов використання гнучких методологій, виконує ключову роль в організації робочих процесів. Він контролює виконання етапів проекту, слідкує за відповідністю результатів вимогам, реагує на ризики та зміни. На рис. 2.4 подано діаграму прецедентів, що відображає додаткові функції менеджера, необхідні в умовах підвищених вимог до управління ризиками.

Рисунок 2.4 – Діаграма прецедентів «Менеджер проєкту»

До основних вимог, що висуваються до засобів автоматизації діяльності менеджера, належать можливості контролю стану проєкту, відстеження якості робіт та забезпечення керування ризиками, що були сформовані експертом та аналітиком.

Системний аналітик відповідає за дослідження предметної області, формування та деталізацію вимог, їх трансформацію відповідно до потреб користувачів та критеріїв якості. Аналітик, інтегруючи запропоновану модель управління ризиками, повинен забезпечити трансформацію потреб користувачів у формальні вимоги, враховуючи їх якісні характеристики, а також забезпечити їх узгодження з визначеними ризиками. На рис. 2.5 зображено основні процеси, автоматизація яких є необхідною для ефективної роботи аналітика.

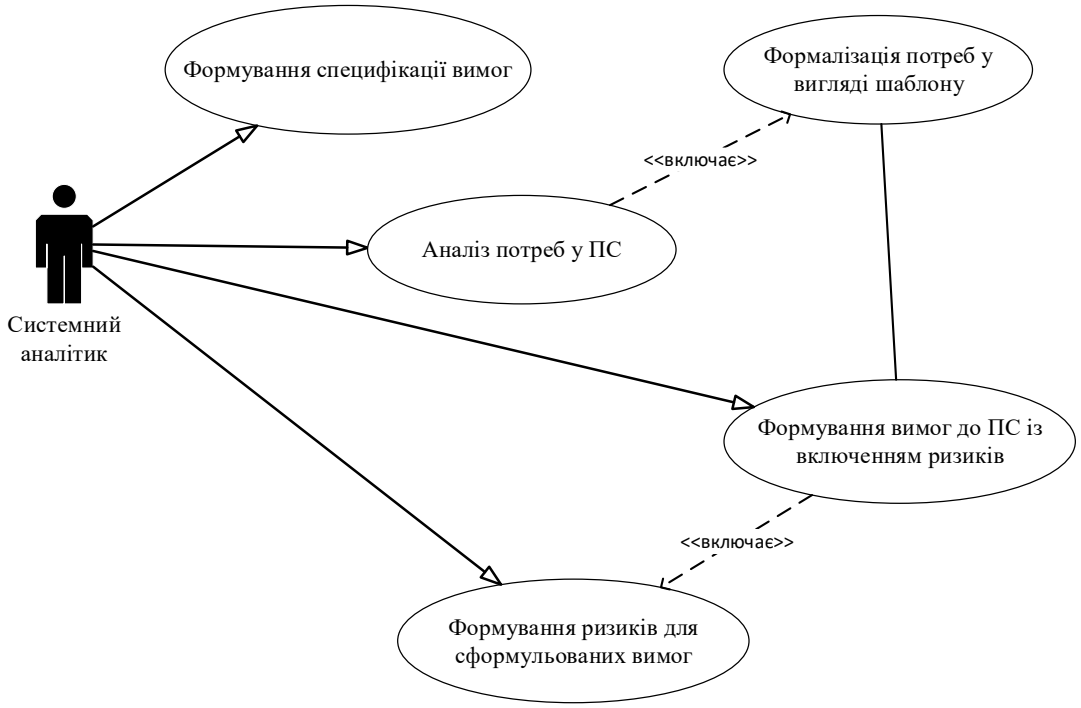

Рисунок 2.5 – Діаграма прецедентів «Системний аналітик»

До компетенції системного аналітика входить формування узгодженого набору вимог до програмного продукту з одночасним врахуванням ризиків, які були визначені експертом. Ключовим завданням є забезпечення їх трасування — тобто простежуваності вимог і ризиків між різними етапами життєвого циклу системи.

Отже, розроблені діаграми прецедентів описують взаємодію всіх учасників процесу розробки, забезпечують системний підхід до управління ризиками відповідно до SEI-моделі та створюють основу для подальшого проектування архітектури програмного засобу, що реалізує розроблений метод.

2.3. Проєктування архітектури програмного засобу на концептуальному рівні

Архітектуру засобу управління, оцінювання та прогнозування ризиків ПЗ у гнучких методологіях на основі моделі SEI запропоновано представити на вигляді багатошарової структури (рис. 2.6).

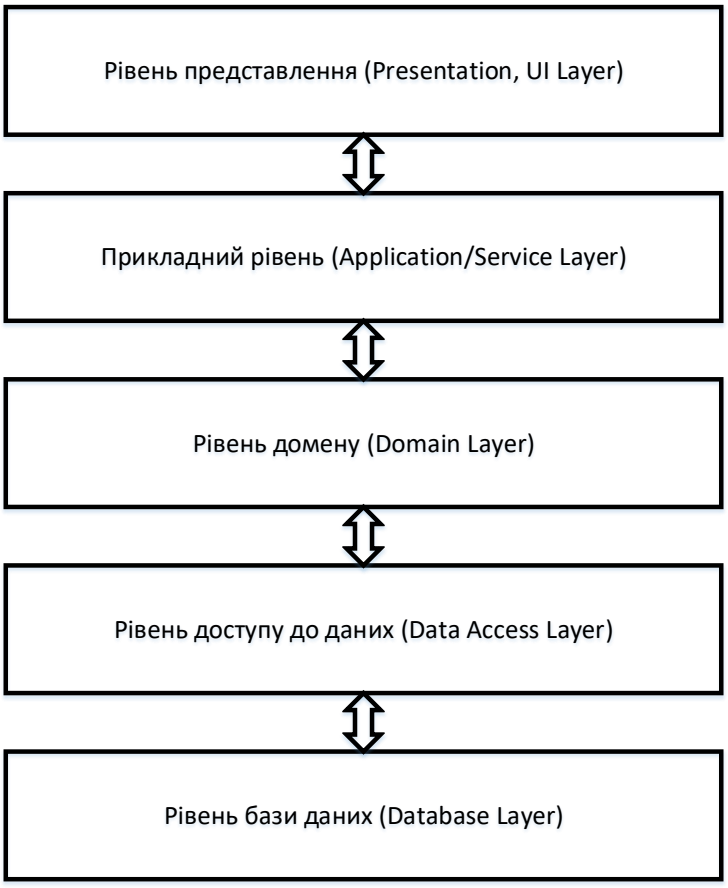

Рисунок 2.6 – Багаторівнева архітектура програмної системи

Основними рівнями архітектури є:

1. Рівень представлення – містить форми користувацького інтерфейсу.
2. Прикладний рівень – реалізує фасади і сервіси, що реалізують сценарії використання.
3. Доменний рівень – модель предметної області: проєкти, спринти, задачі, функціональність, метрики, ризики SEI.
4. Рівень доступу до даних – репозиторії та засоби роботи з СКБД.
5. Рівень бази даних – реалізація у вигляді реляційної БД.

Комунікація між шарами виконується зверху вниз: UI викликає сервіси, сервіси працюють з доменними об’єктами через репозиторії, а репозиторії взаємодіють із БД. Зворотні залежності (знизу вгору) реалізуються тільки через передавання результатів викликів, що забезпечує слабке зв’язування.

Рівень представлення (UI Layer).

На верхньому рівні розташовано пакет UI, який у реалізації відповідає формам/вікнам C#-додатку (WinForms):

- головна форма системи (наприклад, MainForm);
- форма перегляду та вибору проєкту (ProjectView);
- інтерактивна панель оцінювання ризиків (RiskDashboard);
- діалогова форма налаштування метрик (MetricEditorForm) тощо.

З точки зору архітектури цей рівень відображає дані доменної моделі у зручному для користувача вигляді та реагує на дії користувача, наприклад, вибір проєкту, додавання ризику, зміна метрики та ін. Рівень представлення не містить бізнес-логіки, оскільки усі обчислення і перевірки делегуються на нижчі рівні через сервіси. Даний рівень має залежність від пакетів класів прикладного рівня, оскільки форми працюють з відповідними інтерфейсами. Окрім цього, рівень представлення не взаємодіє безпосередньо з доменними класами чи БД.

Прикладний рівень (Service layer).

Прикладний рівень побудований на базі пакета класів services. Він реалізує сценарії використання системи і надає фасадний API для вищих рівнів.

Сервісні інтерфейси.

На основі діаграми класів сервісів виділено такі інтерфейси:

- IProjectService – отримання переліку проєктів та детальної інформації;
- ISprintService – робота зі спринтами певного проєкту;
- ITaskService – доступ до задач, пов’язаних із проєктом чи спринтом;
- IMetricService – конфігурування метрик якості та ризику;
- IRiskService – створення та отримання ризиків, робота з контекстами SEI;
- IRiskAnalysisService – розрахунок експозиції, ранжування та прогнозування ризиків.

Ці інтерфейси формально будуть описані нижче у вигляді UML-моделі і реалізуються конкретними класами (наприклад, ProjectService, RiskService, RiskAnalysisService), які можуть бути розміщені в пакеті application або infrastructure.

На прикладному рівні реалізуються ключові сценарії:

- “Ідентифікація ризиків” – створення нових об’єктів Risk та RiskContext на основі інформації, введеної користувачем у користувацькому інтерейсі;
- “Оцінювання ризиків” – виклик IRiskAnalysisService, який на основі значень метрик і параметрів SEI-моделі обчислює експозицію та інші показники;
- “Моніторинг ризиків” – періодичне оновлення RiskMetric для відстеження динаміки;
- “Налаштування показників” – виклик IMetricService для створення/редагування метрик, що використовуються в системі.

Архітектурно прикладний рівень є посередником між рівнем представлення та доменною моделлю, інкапсулює транзакції, послідовність викликів репозиторіїв і реалізацію алгоритмів SEI.

Доменний рівень (Domain layer)

Доменний рівень є ядром системи і включає пакет domain. До нього входять:

- сутності планування Agile-проєкту: Project, Sprint, Task, SprintTask;
- сутності функціональності й якості: Feature, Metric, FeatureMetric;
- повна модель ризиків SEI: RiskType, SourceRisk, Risk, RiskMetric, RiskContext.

Класи доменного рівня не знають ні про рівень БД, ні про рівень представлення. Вони описують тільки предметну область та правила її цілісності.

Відображення моделі SEI реалізує структуру джерел ризику, типів ризиків, параметрів і контекстів, що дозволяє формалізувати ризики Agile-проєктів у термінах SEI.

Ризики можуть прив’язуватись до різних рівнів процесу: проєкт, спринт, задача, функціональність, що забезпечується через контекстний клас.

Сервіси прикладного рівня працюють саме з цими об’єктами: створюють, модифікують, передають їх між собою й передають у репозиторії для збереження.

Рівень доступу до даних (Data Access layer).

Рівень доступу до даних реалізується пакетом infrastructure і містить репозиторії для роботи з таблицями-довідниками та клас/контекст доступу до БД (DbContext).

Репозиторії реалізують інтерфейси, які використовуються сервісами. Наприклад, RiskService може залежати від інтерфейсу IRiskRepository, а конкретна реалізація цього інтерфейсу працює вже з таблицями Risk, Risk_Type, SourceRisk та RiskContext.Такий поділ дозволяє змінювати технологію зберігання без зміни доменного та прикладного рівнів та легко моделювати й тестувати сервіси, підміняючи реальні репозиторії мок-об’єктами.

Рівень бази даних (Database layer).

Найнижчий рівень відповідає реляційній БД, структура якої буде описана в наступному підрозділі роботи у вигляді ER-діаграми. База даних не взаємодіє безпосередньо з жодним іншим рівнем, окрім Data Access layer. Всі запити, вставки, оновлення виконуються через репозиторії; це дозволяє вести аудит операцій, логи, транзакції та узгоджено реалізовувати алгоритми оцінки ризиків.

Для завершення опису архітектури варто коротко показати основний робочий потік. Користувач на рівні представлення обирає проєкт та ініціює сценарій оцінки ризиків. Відповідна форма звертається до інтерфейсів прикладного рівня, які завантажують із репозиторіїв відповідні доменні об’єкти. Після цього, виконується виклик інтерфейсу аналізу ризиків, який, використовуючи доменні класи, реалізує формули SEI та обчислює критичність ризику. Результати записуються в репозиторії та зберігаються в БД, а обчислені значення рівня ризику повертаються на рівень представлення для візуалізації у вигляді таблиць або звітів. За необхідності користувач може змінювати конфігурацію метрик через відповідний інтерфейс для роботи з метриками, при цьому ці зміни також проходять через доменний рівень та інфраструктурний рівень до БД.

2.4. Побудова діаграм пакетів класів

Діаграма пакетів відображає логічну структуру програмного засобу та розподіл класів між окремими підсистемами. У запропонованій архітектурі виділено такі основні пакети: domain, services, infrastructure, ui та, за потреби, application (рис. 2.7)

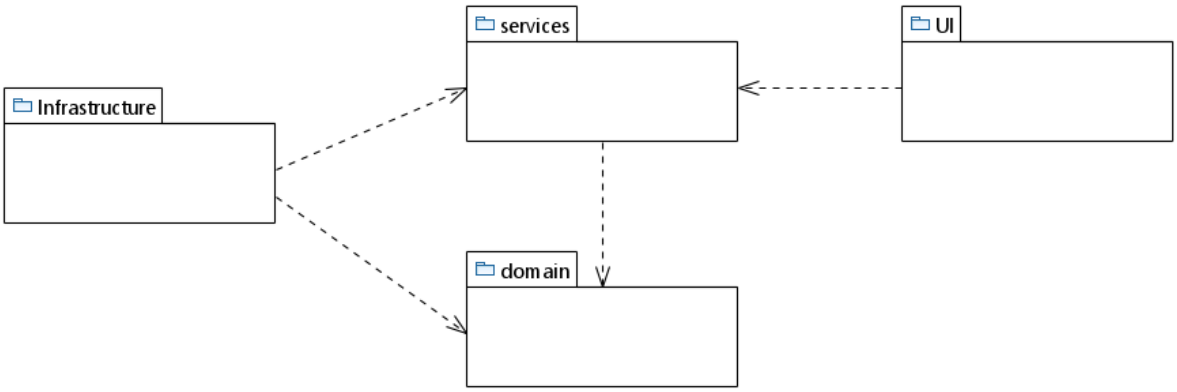

Рисунок 2.7 – Діаграма пакетів класів

Пакет domain містить інформацію про предметну модель: класи, які описують сутності проєктів, спринтів, задач, функціональності, метрик та ризиків SEI. Саме цей пакет реалізує бізнес-логіку в термінах предметної області, оскільки тут зосереджені інваріанти, обмеження, зв’язки між сутностями. Усі інші пакети мають залежності від domain, що відображено на діаграмі пакетів стрілками типу dependency. Таким чином забезпечується централізоване керування еволюцією предметної моделі.

У пакеті services розміщені інтерфейси прикладного рівня. Їхнє завдання полягає у наданні зовнішнім клієнтам високорівневих операцій над доменною моделлю, приховуючи деталі реалізації сховищ, алгоритмів аналізу ризиків та форматів збереження даних. Пакет services має залежність від domain, оскільки використовує типи доменних класів у сигнатурах методів. У реалізації програмного коду інтерфейси пакета services реалізуються окремими класами, що можуть розміщуватися в пакеті infrastructure чи application.

Пакет infrastructure відповідає за технічну реалізацію доступу до даних та інтеграцію з зовнішніми компонентами. Тут розміщуються класи-репозиторії, класи доступу до СКБД, а також адаптери до зовнішніх систем. Пакет має залежність від domain, оскільки репозиторії оперують доменними сутностями, та від services. Такий поділ відповідає підходу «портів і адаптерів», де доменна логіка не залежить від конкретної технології збереження даних.

Пакет application є опційним, але доцільним для більших систем. У ньому можуть розміщуватися фасадні класи та координатори сценаріїв, які оркеструють виклики декількох сервісів для реалізації складного бізнес-сценарію, наприклад, завантажити ризики проєкту чи сформувати звіт. Пакет application залежить від services і domain, але не від infrastructure, що дозволяє легко підмінювати реалізації сервісів, наприклад, при модульному тестуванні.

Пакет ui містить класи графічного інтерфейсу: головну форму застосунку, діалоги керування метриками, вікно перегляду ризиків, форму для відображення діаграм тощо. Класи цього пакета не звертаються безпосередньо до domain або infrastructure, а працюють через інтерфейси пакета services або через фасади пакета application. Така залежність забезпечує чіткий розподіл відповідальностей: інтерфейс відповідає лише за взаємодію з користувачем, тоді як бізнес-логіка і доступ до даних інкапсульовані в нижчих шарах.

Таким чином, діаграма пакетів відображає багатошарову архітектуру програмного засобу, що підтримує принципи модульності, слабкого зв’язування і розширюваності. Завдяки цьому реалізована система керування ризиками на основі моделі SEI може бути адаптована до інших середовищ виконання, наприклад, веб-інтерфейс, сервісна архітектура, без змін у доменній моделі.

2.5. Побудова діаграм класів

Діаграма класів доменної моделі відображає основні поняття предметної області – керування, оцінювання та прогнозування ризиків програмного забезпечення у гнучких (Agile) методологіях на основі адаптованої моделі SEI. На діаграмі виділено декілька груп класів: класи планування та виконання проєкту (Project, Sprint, Task, SprintTask), класи, що описують функціональність і показники якості (Feature, Metric, FeatureMetric), а також класи моделі ризиків SEI (RiskType, SourceRisk, Risk, RiskMetric, RiskContext). На рис. 2.8 показано діаграму класів доменної моделі.

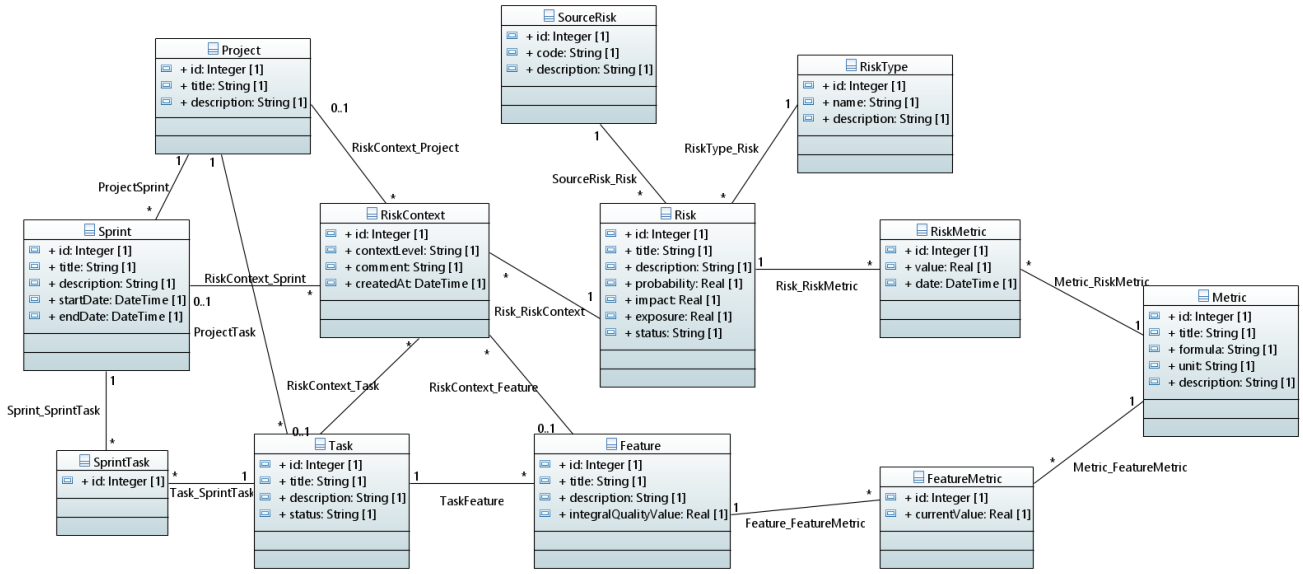

Рисунок 2.8 – Діаграма класів доменної моделі

Клас Project.

Клас Project є кореневою сутністю доменної моделі і описує окремий програмний проєкт, для якого ведеться облік виконання робіт, характеристик якості та ризиків (рис. 2.9).

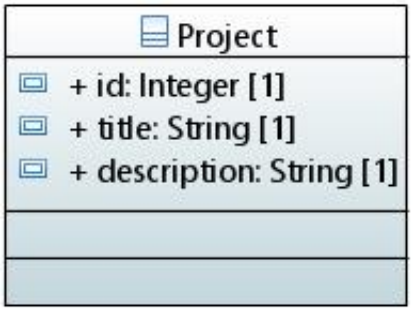

Рисунок 2.9 – Структура класу Project

Атрибути, визначені у класі, зберігають унікальний ідентифікатор, назву та текстовий опис проєкту. Між класом Project та класом Sprint встановлено асоціацію типу «один-до-багатьох»: один проєкт може містити довільну кількість ітерацій-спринтів, тоді як кожний спринт належить рівно одному проєкту. Аналогічна асоціація пов’язує Project з Task – завданнями верхнього рівня, які плануються в рамках проєкту.

Клас Sprint.

Клас Sprint моделює ітерацію розробки в Scrum або іншій Agile-методології (рис. 2.10).

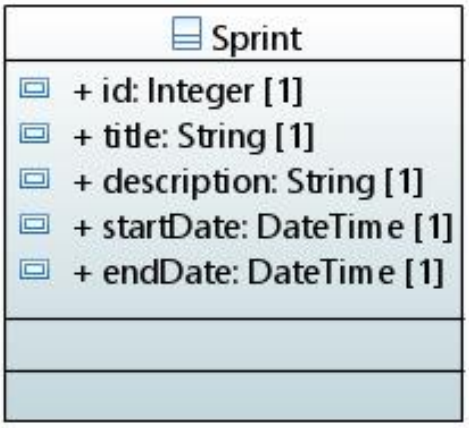

Рисунок 2.10 – Структура класу Sprintt

Структура класу, представлена на рис. 2.10 дозволяє фіксувати часові межі спринту та його короткий опис. Спринт асоційований з класом SprintTask, який реалізує зв’язок «багато-до-багатьох» між спринтами та завданнями: одна і та сама задача може потрапляти в різні спринти (наприклад, при перенесенні або розбитті), а спринт, у свою чергу, містить декілька задач.

Клас Task.

Клас Task описує окреме завдання проєкту (рис. 2.11).

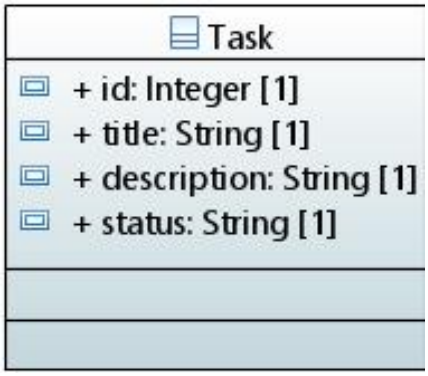

Рисунок 2.11 – Клас Task

Атрибути класу відображають поточний стан виконання проєкту. КласTask пов’язаний з Project таким чином, що кожне завдання належить одному проєкту, а через проміжний клас SprintTask він взаємодіє з однією чи кількома ітераціями. Додатково Task асоційований з Feature, що дозволяє деталізувати реалізовувану функціональність.

Клас SprintTask.

SprintTask представляє собою технічний клас-зв’язок, який реалізує асоціацію «багато-до-багатьох» між Sprint і Task. Даний клас містить атрибут-ідентифікатор та два зовнішні ключі до відповідних сутностей у базі даних. Наявність цього класу спрощує розширення моделі: у разі потреби можна додати атрибути, що описують пріоритет задачі в межах спринту, оцінку трудомісткості тощо.

Клас Feature.

Клас Feature моделює окрему функціональність або вимогу до програмного продукту (рис. 2.12).

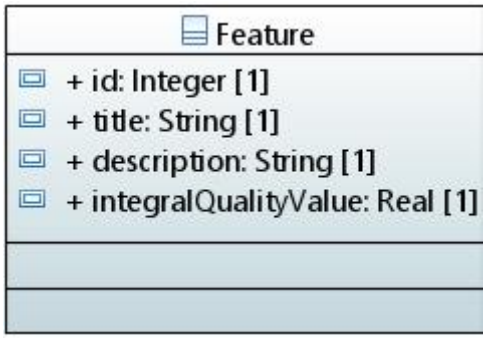

Рисунок 2.12 – Структура класу Feature

Для кожної функціональності ПЗ зберігаються її ідентифікатор, назва та опис, а також інтегральний показник якості, який може використовуватися для агрегування метрик чи розрахунку важливості функціональності. В одному завданні може міститися декілька завдань, що відображено асоціацією «один-до-багатьох».

Клас Metric.

Клас Metric представляє узагальнений опис метрики, яка використовується для формалізованої оцінки характеристик якості або ризику (рис. 2.13).

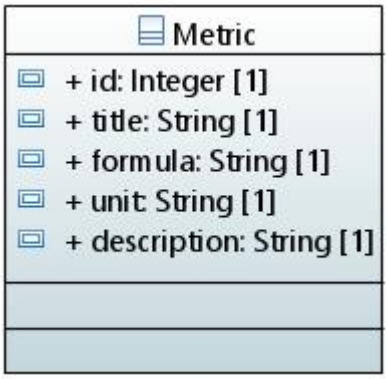

Рисунок 2.13 – Структура класу Metric

Поле formula класу Metric містить текст формули обчислення метрики (наприклад, «кількість реалізованих функцій / загальна кількість функцій»), тоді як unit визначає одиницю вимірювання (відсотки, бали тощо). Це дозволяє гнучко конфігурувати показники без зміни програмного коду.

Клас FeatureMetric.

Клас FeatureMetric реалізує відображення значень метрик на окремі функції (рис. 2.14).

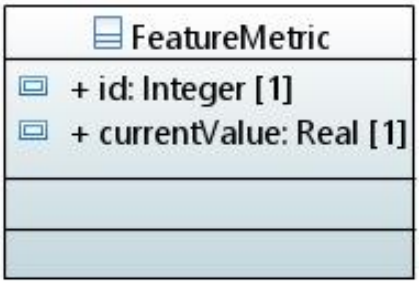

Рисунок 2.14 – Структура класу FeatureMetric

Атрибут currentValue зберігає поточне обчислене значення метрики для конкретної функціональності. Асоціації з класами Feature та Metric мають кратність

1–0..*, що означає: одна функція може бути оцінена за багатьма метриками, одна метрика може застосовуватися до багатьох функцій.

Клас RiskType.

Клас RiskType визначає тип ризику відповідно до адаптованої моделі SEI (рис. 2.15).

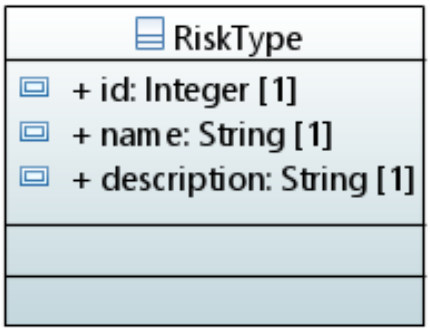

Рисунок 2.15 – Клас RiskType

Асоціація між RiskType та Risk має кратність 1–0..*, тобто один тип ризику може відповідати багатьом конкретним ризикам, але кожний ризик має тільки один тип.

Клас SourceRisk.

Клас SourceRisk відображає «джерело» ризику згідно SEI – це одна з 18 груп потенційних ризиків: функціональність, якість, надійність, бюджет, планування тощо. Структура даного класу представлена на рис. 2.16.

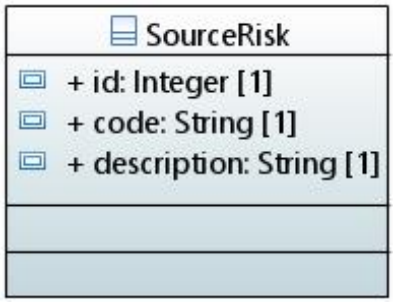

Рисунок 2.16 – Структура класу SourceRisk

Атрибути id, code, description дозволяють зіставляти конкретні ризики з формалізованим набором джерел ризику RS={rs1…rs18}. Асоціація SourceRisk–Risk також має кратність 1–0..*.

Клас Risk.

Клас Risk – центральний елемент моделі ризиків (рис. 2.17). Він містить опис конкретного ризику.

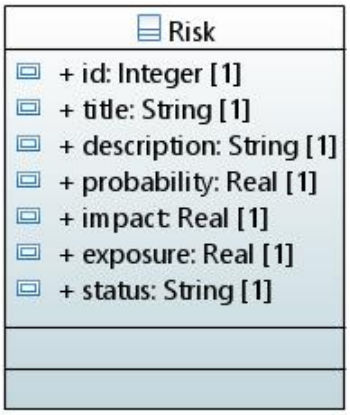

Рисунок 2.17 – Структура класу Risk

Ймовірність і вплив відповідають класичній SEI-моделі, де експозиція (Risk Exposure) обчислюється як добуток ймовірності на розмір втрат. Атрибут status дозволяє відстежувати життєвий цикл ризику (ідентифікований, в роботі, пом’якшений, закритий).

Клас RiskMetric.

Клас RiskMetric gризначений для зберігання історії зміни кількісних показників ризику (рис. 2.18).

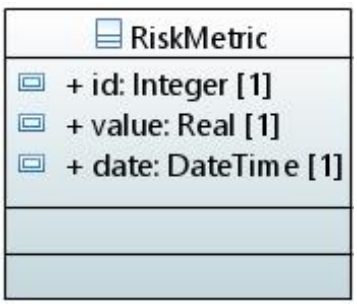

Рисунок 2.18 – Структура класу RiskMetric

Атрибути value та date фіксують значення обраної метрики для конкретного ризику в певний момент часу. Зв’язки Risk – RiskMetric та Metric – RiskMetric мають кратність 1–0..*, що дозволяє накопичувати часові ряди значень різних метрик для кожного ризику.

Клас RiskContex.

Клас RiskContext (рис. 2.19) задає контекст прояву ризику у межах проєкту і вказує, до яких елементів Agile-процесу цей ризик відноситься.

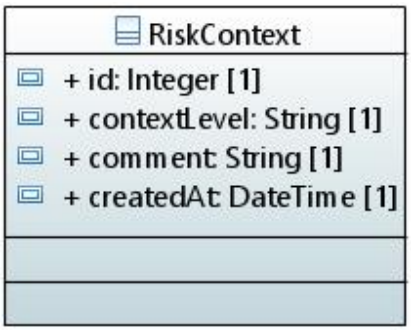

Рисунок 2.19 – Клас RiskContext

Атрибути класу описують рівень контексту ризику: проєкт, спринт, задача, функція, зберігають пояснювальний коментар та час створення. Клас має обов’язкову асоціацію з Risk, оскільки кожний контекст описує певний ризик і необов’язкові асоціації (0..1) з Project, Sprint, Task та Feature.

Таким чином, один ризик може мати декілька контекстів прояву: наприклад, ризик «недостатня продуктивність» може бути прив’язаний як до всього проєкту, так і до конкретної функції або спринту.

У сукупності діаграма класів доменної моделі відображає всі сутності, необхідні для реалізації технології аналізу та керування ризиками ПЗ в Agile-процесі: від ієрархії проєкт–спринт–задача–функція до повної структури моделі SEI з типами, джерелами, кількісними метриками і контекстами ризиків.

Діаграма класів сервісного рівня.

Окрема діаграма класів сервісів описує інтерфейсний рівень доступу до доменної моделі. Вона відображає те, як прикладні сценарії взаємодіють із класами domain через набір сервісних інтерфейсів. Такий підхід відповідає принципам

розділення відповідальностей і спрощує тестування та заміну реалізацій. Діаграма інтерфейсів класів показана на рис. 2.20.

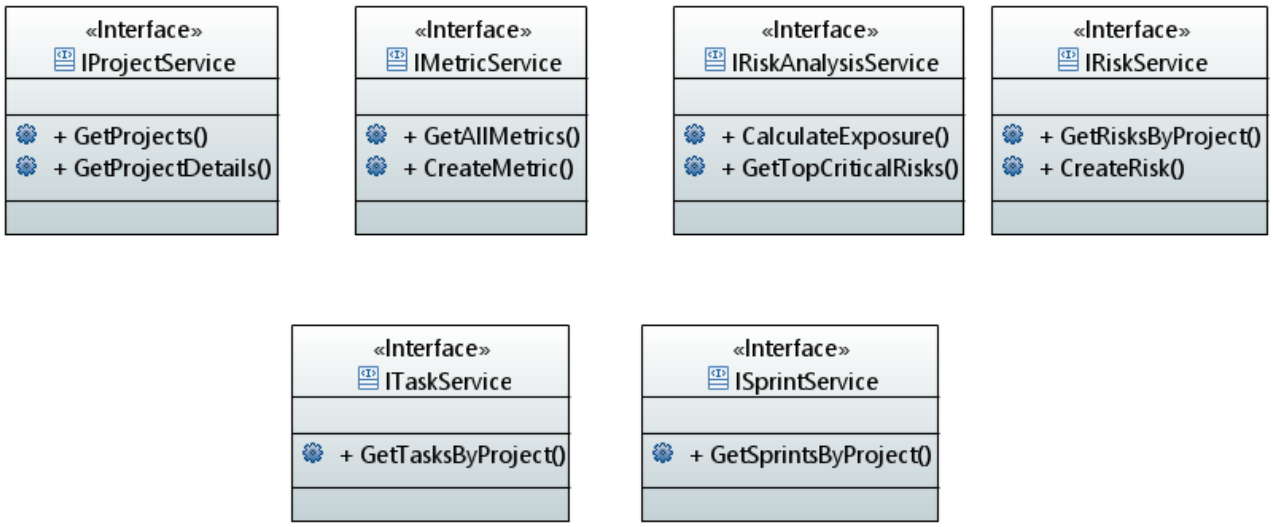

Рисунок 2.20 – Діаграма інтерфейсів класів прикладного рівня

Інтерфейс IProjectService.

Інтерфейс IProjectService (рис. 2.21) надає методи для роботи з проєктами: GetProjects() повертає перелік усіх зареєстрованих проєктів, а GetProjectDetails() – детальну інформацію про обраний проєкт, включно з пов’язаними спринтами, задачами, метриками та ризиками.

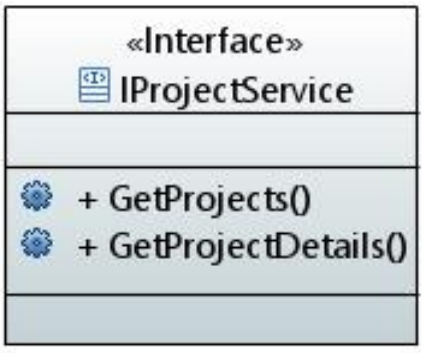

Рисунок 2.21 – Інтерфейс IProjectService

Інтерфейс використовується як у графічному інтерфейсі користувача, так і у сервісах аналізу ризиків.

Інтерфейс ISprintService.

Інтерфейс ISprintService (рис. 2.22) інкапсулює операції над спринтами: основний метод GetSprintsByProject() повертає список ітерацій для заданого проєкту.

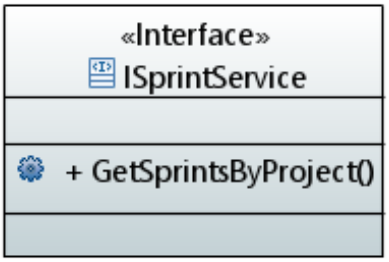

Рисунок 2.22 – Інтерфейс ISprintService

Потенційно сервіс може бути розширений можливостями створення, редагування й закриття спринтів, однак в рамках даної системи акцент зроблено на аналітичних операціях.

Інтерфейс ITaskService.

Інтерфейс ITaskService відповідає за роботу із задачами, його структуру показано на рис. 2.23.

Рисунок 2.23 – Інтерфейс ITaskService

Метод GetTasksByProject() повертає перелік задач у межах вибраного проєкту з урахуванням їхнього статусу. Через цей сервіс користувач може отримати деталізовану інформацію про те, які роботи виконуються в кожному спринті, і зіставити їх з відповідними ризиками.

Інтерфейс IMetricService.

Інтерфейс IMetricService забезпечує управління метриками якості та ризику (рис. 2.24).

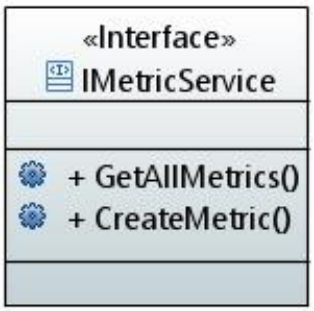

Рисунок 2.24 – Інтерфейс IMetricService

Операція GetAllMetrics() повертає список усіх визначених метрик, а CreateMetric() дозволяє зареєструвати нову метрику разом з формулою та описом. Цей сервіс використовується при конфігуруванні системи – наприклад, при додаванні користувацьких показників для окремого проєкту.

Інтерфейс IRiskService.

Інтерфейс IRiskService реалізує CRUD-функціональність для ризиків, структура якого наведена на рис. 2.25.

Операція GetRisksByProject() використовується для завантаження переліку всіх ризиків, пов’язаних із конкретним проєктом (через контексти RiskContext).

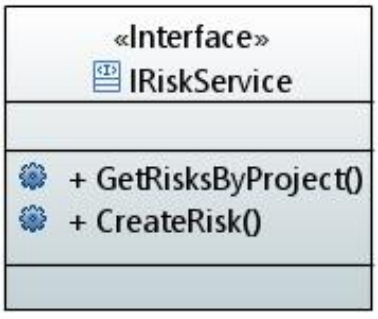

Рисунок 2.25 – Структура IRiskService

Метод CreateRisk() дає змогу додати новий ризик із заповненням основних атрибутів моделі SEI, зокрема, типу, джерела, ймовірності, впливу тощо.

У перспективі інтерфейс може бути доповнений операціями оновлення експозиції, зміни статусу, фіксації заходів реагування.

Інтерфейс IRiskAnalysisService.

Інтерфейс IRiskAnalysisService містить «інтелектуальну» логіку аналізу ризиків (рис. 2.26).

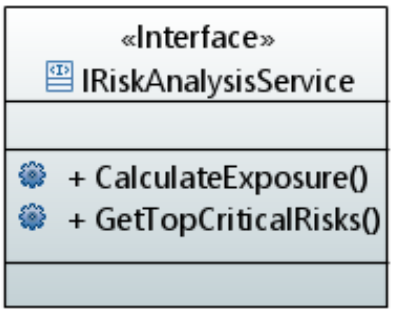

Рисунок 2.26 – Інтерфейс IRiskAnalysisService

Метод CalculateExposure() на основі списку ризиків та значень відповідних метрик виконує розрахунок рівня критичності ризиків та інших похідних показників. Операція GetTopCriticalRisks() повертає підмножину найбільш критичних ризиків за заданим критерієм.

Сукупність сервісних інтерфейсів формує чітко визначений API системи. Класи графічного інтерфейсу, а також можливі зовнішні компоненти (наприклад, модуль експорту звітів) працюють лише через ці інтерфейси, не звертаючись безпосередньо до доменної моделі чи до бази даних. Це забезпечує слабке зв’язування та дає можливість заміни реалізацій сервісів без зміни загальної архітектури.

2.6. Проектування структури бази даних

Для зберігання даних у роботі створено реляційну базу даних, ER-діаграму якої представлено на рис. 2.27. Середовище для реалізації БД – MS SQL Server.

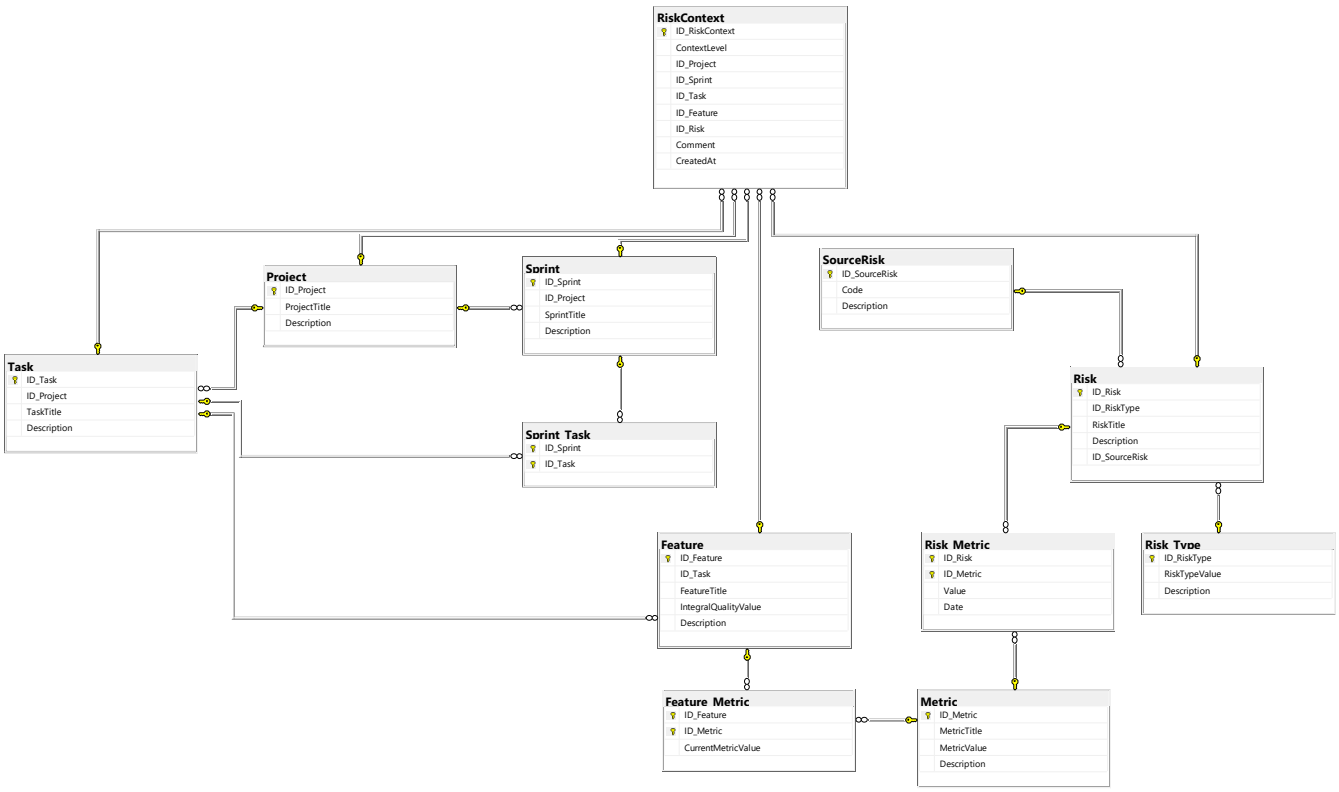

Рисунок 2.27 – ER-діаграма БД

ER-діаграма бази даних [20-23] конкретизує доменну модель у вигляді реляційної структури. Вона містить таблиці, що відповідають класам доменної моделі, та проміжні таблиці, які реалізують зв’язки «багато-до-багатьох» і контексти ризиків. Нижче наведено призначення ключових таблиць та їхніх полів.

Таблиця Project зберігає загальну інформацію про програмні проєкти. Структура таблиці включає поля: ID_Project (цілочисельний первинний ключ), ProjectTitle (назва проєкту), Description (текстовий опис). На ER-діаграмі ця таблиця пов’язана зв’язками «один-до-багатьох» з таблицями Sprint, Task і через таблицю RiskContext з таблицею Risk. Таким чином, проєкт виступає кореневою сутністю, від якої успадковується контекст більшості даних.

Таблиця Sprint реалізує збереження даних про ітерації розробки проєкту. Основні поля: ID_Sprint (PK), ID_Project (FK на Project), SprintTitle, Description. Наявність зовнішнього ключа забезпечує зв’язок спринту з відповідним проєктом. У деяких реалізаціях до таблиці можуть бути додані поля StartDate та EndDate, які прямо відповідають атрибутам доменного класу. У зв’язці з таблицею Sprint_Task ця таблиця дозволяє відстежувати, які задачі виконуються в кожній ітерації.

Таблиця Task містить записи про задачі, що плануються й виконуються в межах проєктів. Структура: ID_Task (PK), ID_Project (FK), TaskTitle, Status, Description. Через таблицю Sprint_Task кожне завдання може бути прив’язане до одного чи кількох спринтів. Це дозволяє реалізувати перенесення незавершених задач між ітераціями та аналізувати розподіл робіт.

Таблиця Sprint_Task є проміжною таблицею для зв’язку «багато-до-багатьох» між Sprint та Task. Вона містить поля ID_Sprint та ID_Task, які утворюють складений первинний ключ та одночасно є зовнішніми ключами на відповідні таблиці. За потреби до цієї таблиці можуть бути додані додаткові атрибути, однак базова модель обмежується зберіганням самого факту належності.

Таблиця Feature відповідає за збереження даних про функціональність, яка реалізується у програмного продукті, який розробляється. Основні поля: ID_Feature (PK), ID_Task (FK), FeatureTitle, Description, IntegralQualityValue. Зовнішній ключ ID_Task забезпечує зв’язок з таблицею Task: одна задача може містити кілька функцій. Інтегральне значення якості використовується для зберігання агрегованих показників, отриманих на основі метрик, і може застосовуватись при пріоритизації backlog’у.

Таблиця Metric містить довідникову інформацію про метрики: ID_Metric (PK), MetricTitle, MetricValue, Description. У деяких варіантах структура може доповнюватися полями Formula та Unit, які зберігають формулу обчислення і одиницю виміру. На ER-діаграмі Metric пов’язана з таблицями Feature_Metric та Risk_Metric, що дозволяє використовувати одну й ту ж метрику як для оцінювання функціональності, так і для оцінки ризиків.

Таблиця Feature_Metric є проміжною таблицею, яка фіксує значення конкретної метрики для конкретної функції. Вона має поля ID_Feature (FK на Feature), ID_Metric (FK на Metric) та CurrentMetricValue (значення метрики). Складений ключ (ID_Feature, ID_Metric) гарантує, що для пари «функція–метрика» існує не більше одного запису. Така структура дозволяє додавати нові метрики без зміни схеми бази даних.

Таблиця Risk_Type є довідником типів ризиків. Структура: ID_RiskType (PK), RiskTypeValue (коротке позначення), Description. Типи відповідають групам ризиків у моделі SEI. Зв’язок з таблицею Risk має вид «один-до-багатьох»: один тип може бути використаний у багатьох ризиках.

Таблиця SourceRisk реалізує множину джерел ризику, яка описана в моделі SEI. Кожний запис узагальнює певну категорію ризиків – характеристики функціональності, якості, надійності, бюджетні обмеження, планування тощо. Через зовнішній ключ ID_SourceRisk таблиця Risk прив’язує кожен конкретний ризик до одного з визначених джерел.

Таблиця Risk є центральною у моделі управління ризиками. Вона містить поля: ID_Risk (PK), ID_RiskType (FK), ID_SourceRisk (FK), RiskTitle, Description, Probability, Impact, Exposure, Status. Значення Probability та Impact задаються у діапазоні [0;1] або у відсотках, а Exposure може обчислюватися автоматично на основі цих полів та пов’язаних метрик. Зовнішні ключі забезпечують узгодженість даних з довідниками типів та джерел ризиків.

Таблиця Risk_Metric служить для зберігання динаміки числових показників ризику. Основні атрибути: ID_Risk (FK), ID_Metric (FK), Value, Date. Як і у випадку з Feature_Metric, складений ключ (ID_Risk, ID_Metric, Date) дозволяє фіксувати значення однієї й тієї ж метрики в різні моменти часу. Це дає змогу будувати графіки зміни ризику, відстежувати ефективність застосованих заходів реагування тощо.

Нарешті, таблиця RiskContext реалізує багатовимірний контекст прояву ризику. Вона містить поля: ID_RiskContext (PK), ContextLevel, Comment, CreatedAt та набір зовнішніх ключів – ID_Risk (обов’язковий), ID_Project, ID_Sprint, ID_Task, ID_Feature (необов’язкові). Така структура дозволяє описати, у межах якого саме елемента Agile-процесу спостерігається даний ризик: для всього проєкту, для конкретного спринту, задачі або навіть окремої функції. Один ризик може мати декілька записів у RiskContext, що відповідає різним сценаріям прояву.

Описана ER-модель забезпечує нормалізоване зберігання даних про проєкти, планування робіт, функціональність, метрики та ризики. Вона безпосередньо

відображає сутності та зв’язки, представлені на діаграмі класів, і виступає технічною основою для реалізації алгоритмів аналізу, оцінювання та прогнозування ризиків програмного забезпечення у гнучких методологіях розробки.

2.7. Висновки до другого розділу

1. Розроблено та теоретично обґрунтовано метод управління ризиками програмного забезпечення, який базується на інтеграції моделі ризиків SEI з моделями якості стандарту ISO/IEC 25010 та адаптований до використання у гнучких (Agile) методологіях розробки. Це дало змогу пов’язати ризики з вимогами до якості програмного продукту та забезпечити їх системне врахування на всіх етапах життєвого циклу ПЗ.

2. Проведено аналіз предметної області та ролей учасників процесу розробки, у результаті якого визначено функціональні обов’язки системного аналітика, експерта з ризиків і менеджера проєкту в контексті Agile-процесів. Побудовані UML-діаграми прецедентів формалізують сценарії взаємодії користувачів із системою та створюють основу для автоматизації процесів ідентифікації, аналізу та моніторингу ризиків.

3. Запропоновано багаторівневу архітектуру програмного засобу управління ризиками, яка включає рівні представлення, прикладної логіки, доменної моделі, доступу до даних і бази даних. Такий підхід забезпечує розділення відповідальностей, слабке зв’язування компонентів та можливість подальшого масштабування або інтеграції системи з іншими програмними рішеннями.

4. Розроблено діаграми пакетів і класів, що відображають логічну структуру програмного засобу та формалізують ключові сутності предметної області: проєкти, спринти, задачі, функціональність, метрики якості та повну модель ризиків SEI. Це дало змогу чітко визначити взаємозв’язки між елементами Agile-

процесу та ризиками, а також створити основу для реалізації алгоритмів оцінювання і прогнозування ризиків.

5. Спроєктовано структуру реляційної бази даних у вигляді ER-діаграми, яка узгоджується з доменною моделлю та забезпечує нормалізоване зберігання даних про проєкти, планування робіт, вимоги, метрики та ризики. Запропонована модель БД дозволяє накопичувати історію змін ризиків, аналізувати їх динаміку та використовувати ці дані для підтримки прийняття управлінських рішень у подальших проєктах.

6. Сформовано цілісну концептуальну та архітектурну основу програмного засобу, що реалізує метод управління ризиками на основі моделі SEI в Agile-середовищі. Це створює передумови для подальшої реалізації програмного продукту, його практичної апробації та розширення функціональності шляхом автоматизованого аналізу, моніторингу й прогнозування ризиків програмного забезпечення.

3 ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ЗАСТОСУВАННЯ МЕТОДУ І ЗАСОБУ УПРАВЛІННЯ РИЗИКАМИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

У третьому розділі проведено тестування та експериментальну перевірку розробленого програмного засобу управління ризиками. У результаті виконання процедур ручного тестування підтверджено його відповідність визначеним функціональним вимогам. В якості експерименту та на основі експертних технологій проведено ідентифікацію та ранжування ризиків, що можуть виникнути в процесі розробки web-орієнтованого програмного продукт.

3.1. Тестування засобу управління ризиками до програмного забезпечення

Користувацький інтерфейс програмного засобу, призначеного для підвищення результативності виконання програмних проєктів і підтримки процесів управління ризиками, має відповідати вимогам простоти, інтуїтивності та зручності використання. З цією метою під час проєктування інтерфейсу особливу увагу було приділено принципам ергономіки та узгодженості елементів керування. Загальний вигляд головного вікна програмної системи наведено на рис. 3.1.

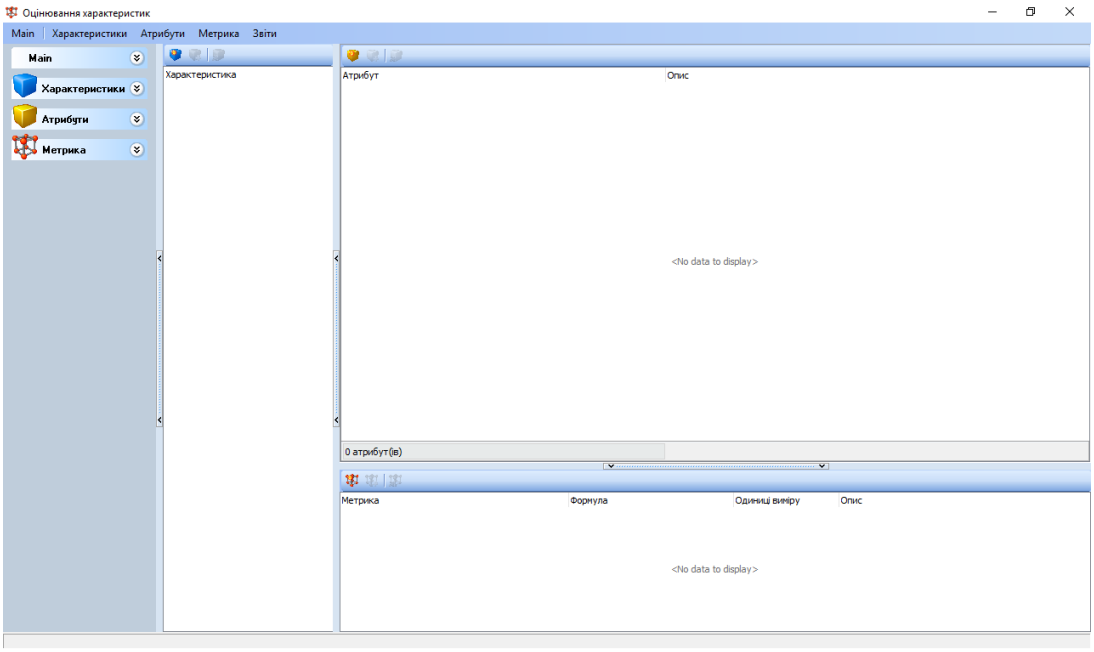

Рисунок 3.1 – Форма головного вікна

Головне вікно, представлене на рис. 3.4, поєднує класичне верхнє меню та навігаційну панель, розташовану зліва, що забезпечує зручний доступ до основних функцій формування вимог і ризиків програмного продукту. Структуру основного меню «Main» подано на рис. 3.2.

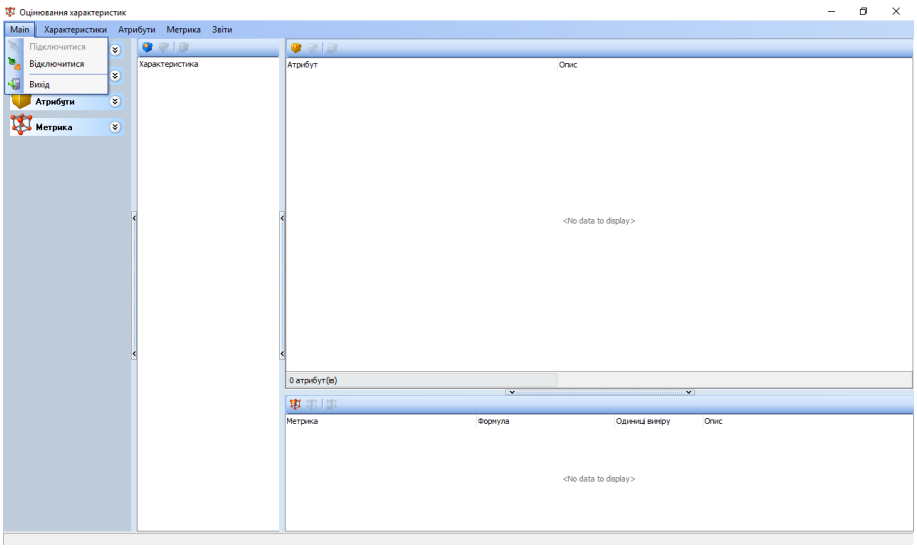

Рисунок 3.2 – Меню «Main»

До основних пунктів меню «Main» належать команди підключення та відключення бази даних, у якій зберігаються відомості про вимоги й ризики програмних складових, а також функція завершення роботи із програмним засобом.

На рис. 3.3 наведено структуру меню «Характеристики», яке призначене для роботи з узагальненими вимогами до програмної системи на рівні її архітектури. За допомогою відповідних пунктів меню користувач може створювати, змінювати та видаляти характеристики якості. Вікно додавання нової характеристики якості показано на рис. 3.4.

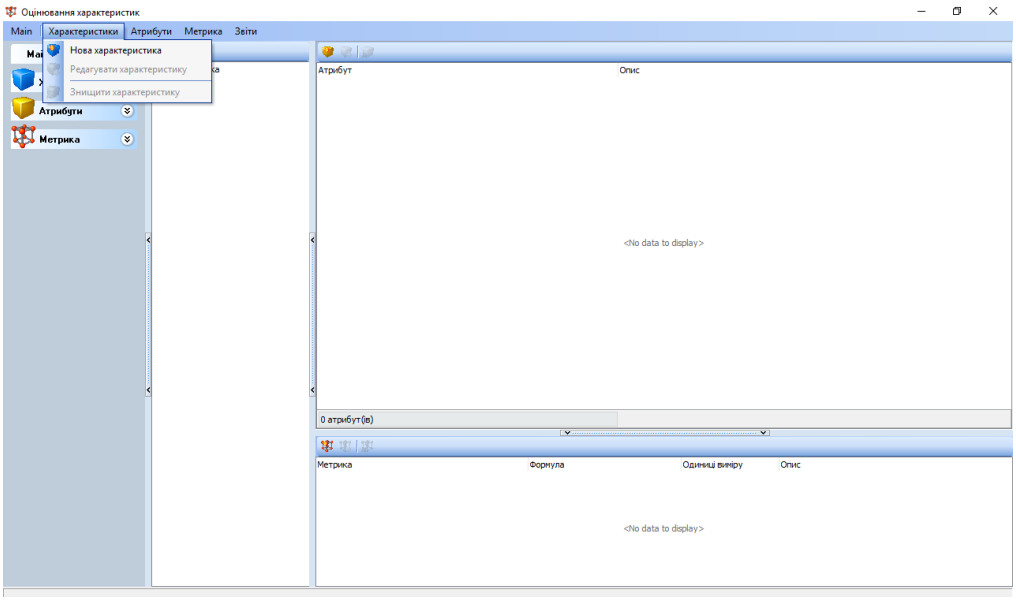

Рисунок 3.3 – Меню створення характеристик

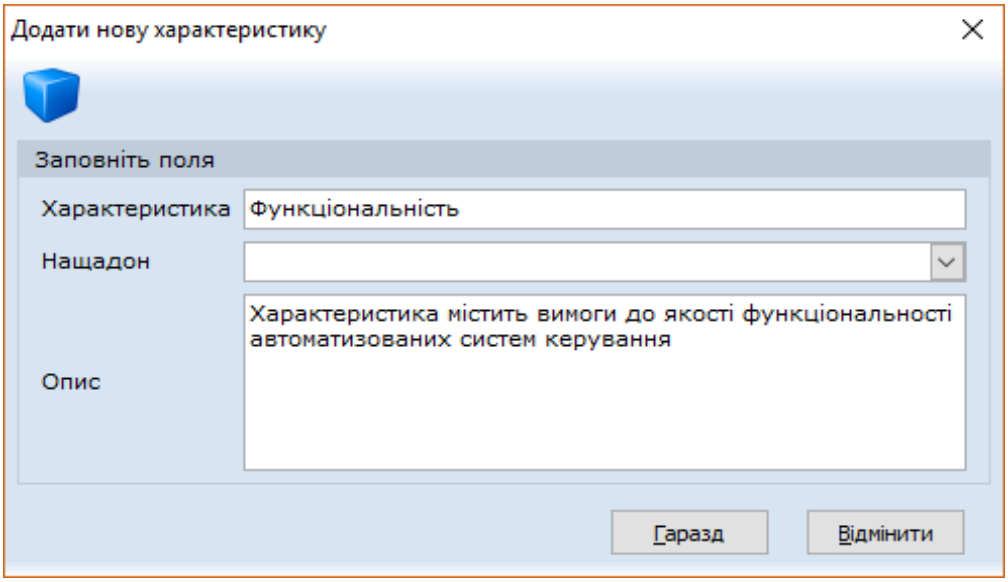

Рисунок 3.4 – Діалогове вікно формування характеристики

Меню «Атрибути» використовується для опрацювання локальних вимог, тобто конкретних вимог до програмної системи. Воно забезпечує можливість створення, редагування та видалення атрибутів. При цьому додавання або модифікація атрибута можлива лише після попереднього створення відповідної характеристики, до якої даний атрибут віднесений експертом. Інтерфейс меню роботи з атрибутами наведено на рис. 3.5, а форма додавання та редагування атрибутів – на рис. 3.6.

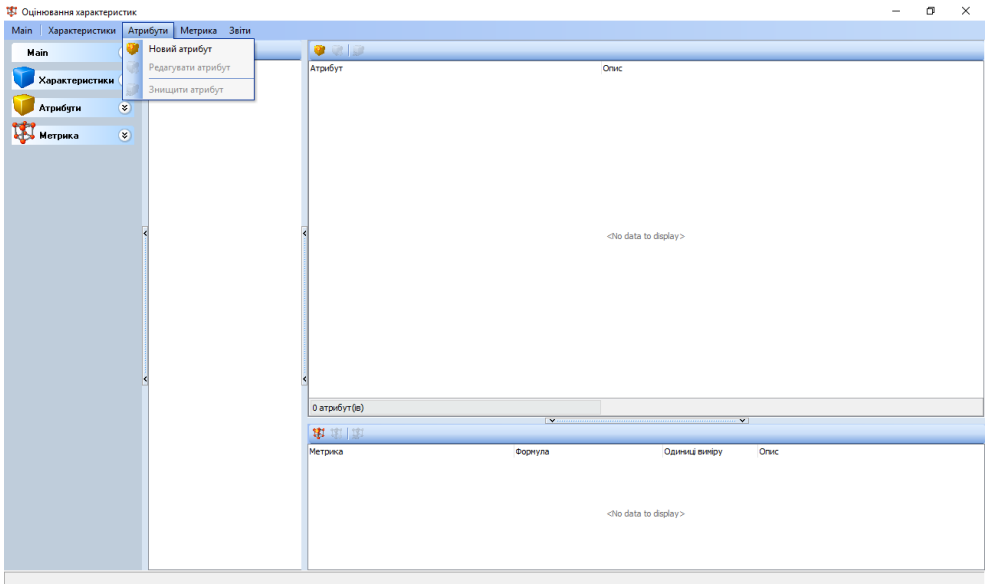

Рисунок 3.5 – Вікно додавання нового атрибуту характеристики

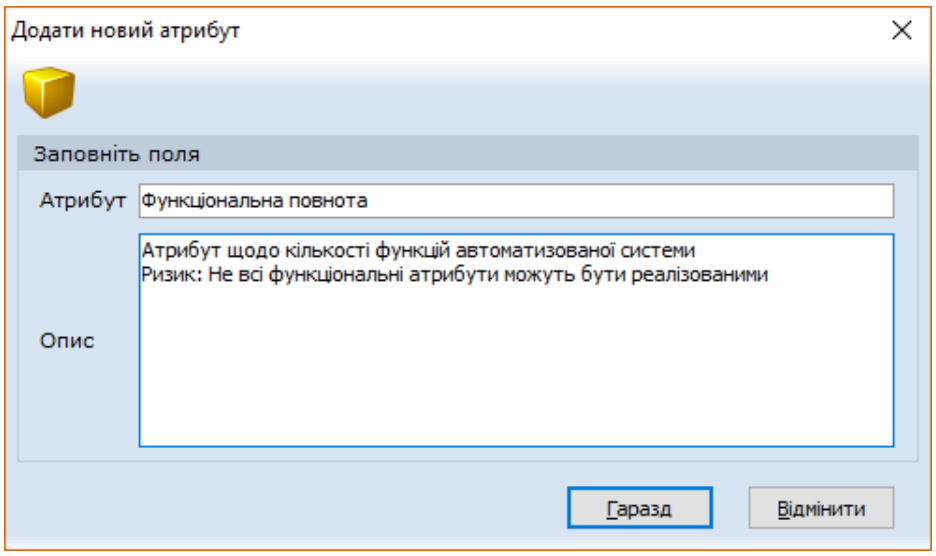

Рисунок 3.6 – Діалогова форма для роботи з атрибутами

Оцінювання ступеня реалізації атрибутів та перевірка їх відповідності встановленим вимогам здійснюється за допомогою меню «Метрика». Дане меню дозволяє створювати, змінювати та видаляти метрики, що характеризують реалізацію відповідних атрибутів. Слід підкреслити, що створення метрики без попередньо визначеної вимоги є неможливим, оскільки метрика відображає кількісну оцінку ступеня реалізації конкретного атрибуту. Структуру меню «Метрика» наведено на рис. 3.7, а вікно додавання та редагування метрик – на рис. 3.8.

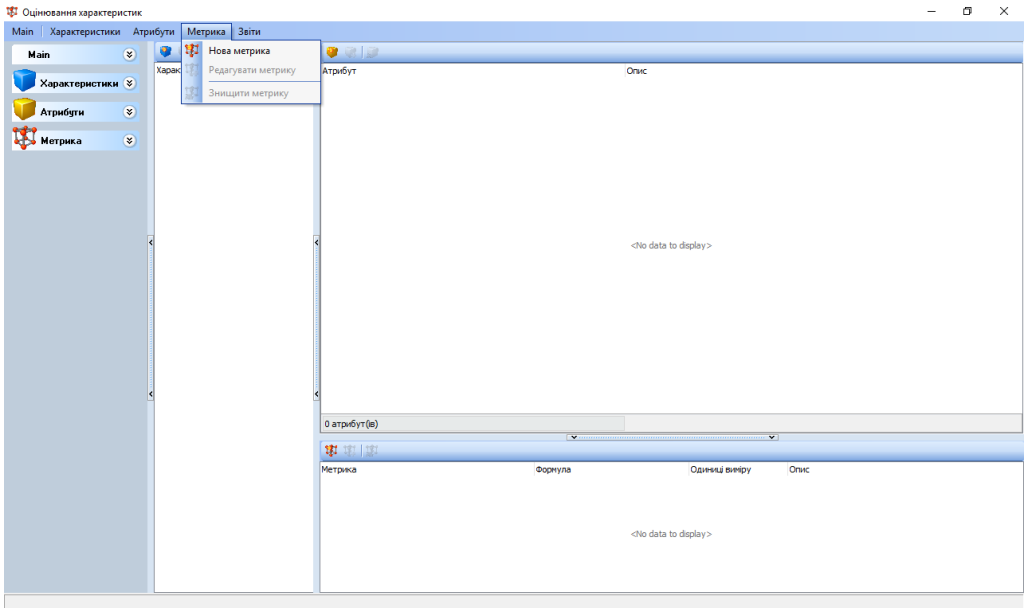

Рисунок 3.7 – Меню для роботи з метриками

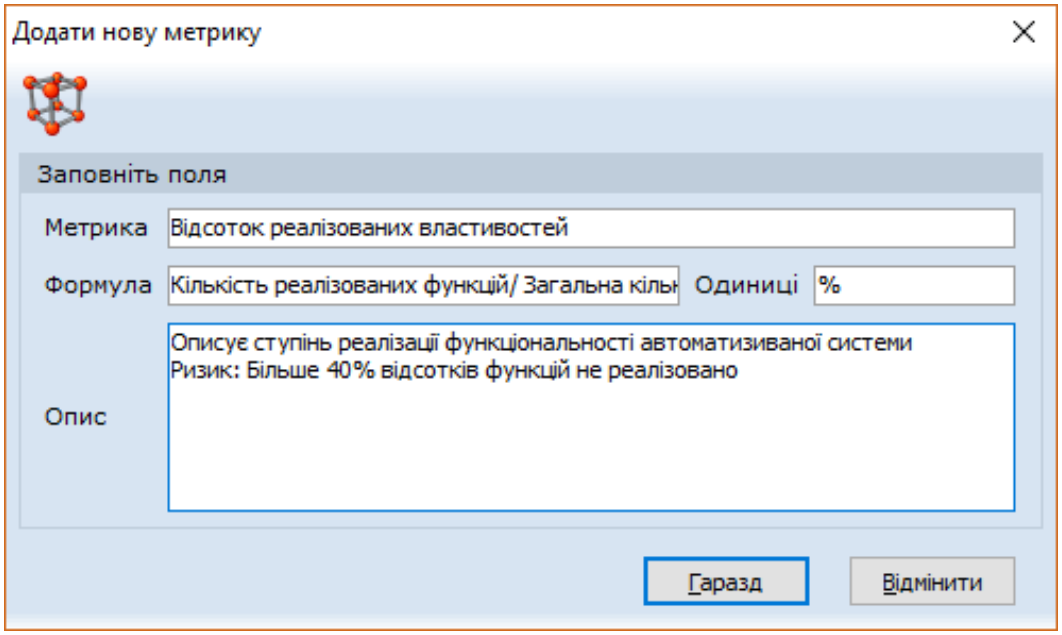

Рисунок 3.8 – Реадагування метрик

Після створення характеристики «Функціональність», відповідного атрибуту «Функціональна повнота» та метрики «Відсоток реалізованих функцій» система автоматично формує вікно з тестовими даними, приклад якого наведено на рис. 3.9.

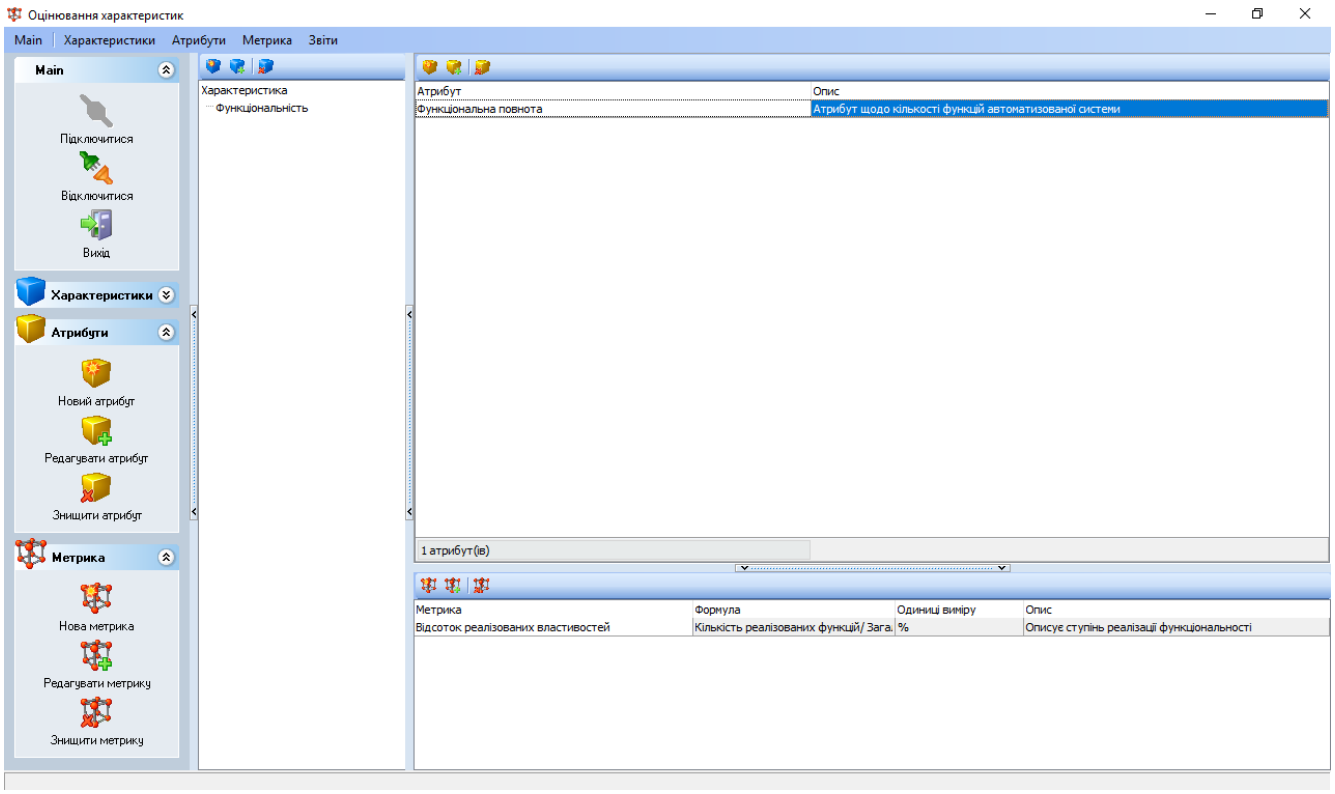

Рисунок 3.9 – Інтерфейс засобу з тестовими даними

Редагування тестових значень здійснюється за допомогою панелі інструментів, розташованої у лівій частині вікна. Повна ієрархічна структура моделей якості та їх характеристик відображається у вигляді дерева, яке розміщене між панеллю інструментів і табличним представленням атрибутів та метрик програмного продукту.

Залежно від ролі користувача дана форма забезпечує можливість формування критеріїв вимог як до програмних систем, так і до процесів їх реалізації. У зв’язку з цим актуальним завданням є створення інтерфейсу для гнучкого формування звітної документації, зокрема специфікацій вимог до програмного забезпечення та звітів про ступінь їх реалізації на різних етапах життєвого циклу. Відповідні форми налаштування параметрів і генерації документації наведено на рис. 3.10 та рис. 3.11.

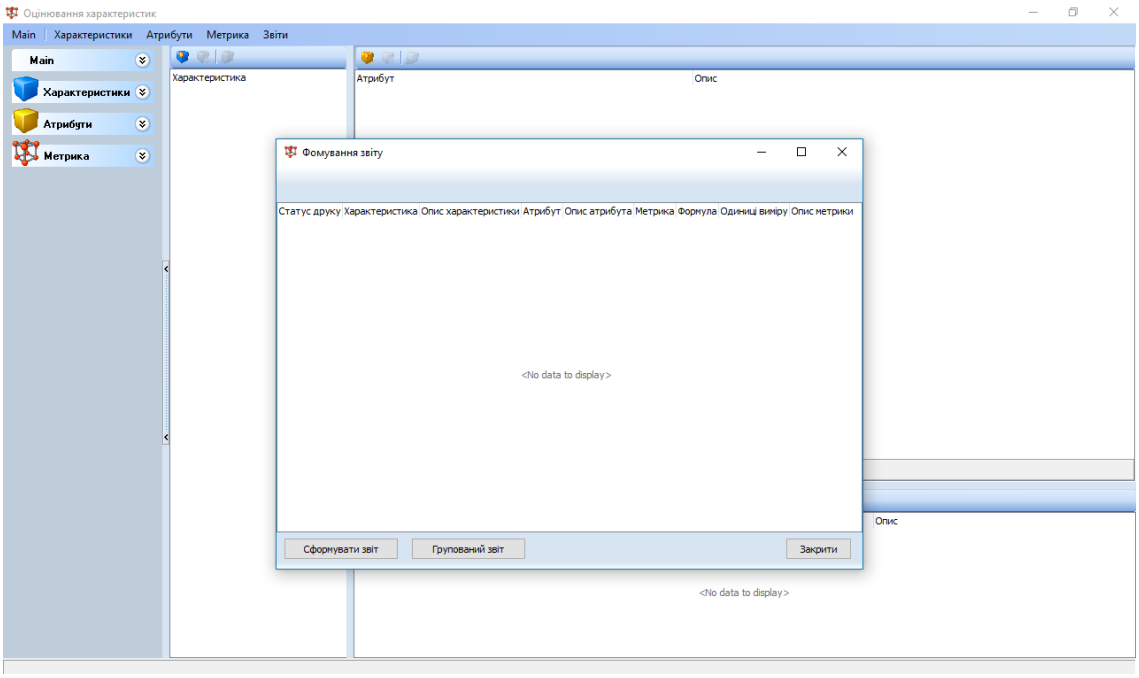

Рисунок 3.10 – Вікно налаштувань при формуванні звіту

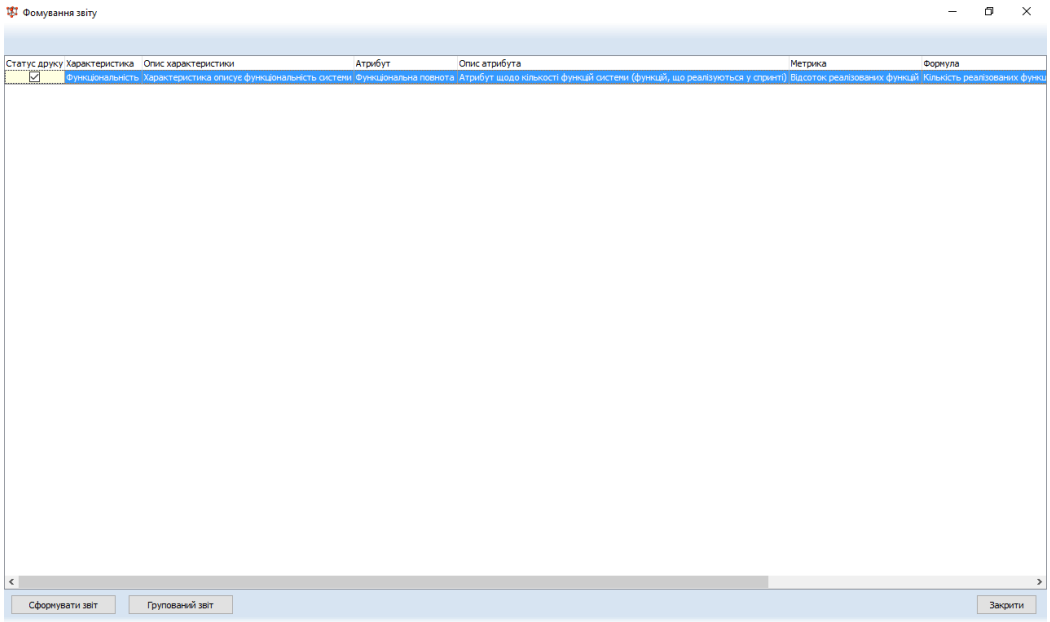

Рисунок 3.11 – Генерація вмісту звіту

Отже, в результаті проведеного тестування у відповідності до функціональних вимог до програмного засобу підтримки процесів розробки програмного забезпечення з урахуванням управління ризиками, проєктування його архітектури та розробки користувацьких інтерфейсів було створено повнофункціональний програмний продукт, орієнтований на автоматизацію

процесів підвищення якості реалізації програмних проєктів. Подальшим напрямом розвитку системи є створення прикладного програмного інтерфейсу (API) для інтеграції з іншими інструментами управління процесами розробки програмного забезпечення.

3.2. Визначення критичного шляху та ідентифікація ризиків

З метою практичної перевірки ефективності запропонованого підходу до виявлення та управління ризиками розглянуто приклад реалізації програмного проєкту зі створення web-орієнтованого сервісу. Проєкт характеризується типовою структурою робіт, що охоплює всі ключові етапи життєвого циклу програмного забезпечення та може бути використана як еталонна модель для подібних розробок.

Для формалізації проєктної діяльності виконано деталізацію робіт із застосуванням ієрархічної декомпозиції, яка дозволяє встановити логічні та часові залежності між окремими етапами розробки. Сформована структура включає процеси аналізу та погодження вимог, проєктування інтерфейсу, розробку функціональної логіки, тестування, розгортання та подальший супровід програмного продукту [30].

Для практичної перевірки та апробації запропонованого методу ідентифікації й управління ризиками розглянемо приклад реалізації невеликого програмного проєкту, спрямованого на створення web-орієнтованого сервісу:

1. Формування вимог.

1.1. Аналіз і збір вимог замовника.

1.2. Опрацювання вимог профільними спеціалістами.

1.3. Погодження та затвердження вимог клієнтом.

2. Розробка графічного дизайну.

2.1. Створення корпоративного стилю.

2.2. Проєктування дизайн-макета.

3. Верстка головної сторінки та підсторінок.

3.1. Логічне структурування макета.

3.2. Деталізація елементів макета.

3.3. Верстка відповідно до стандартів W3C.

3.4. Перевірка коректності та валідація.

4. Кросбраузерне тестування.

5. Формування структури сайту.

5.1. Побудова базової ієрархії.

5.2. Визначення зв’язків між сторінками.

5.3. Реалізація структури.

6. Наповнення сайту графічним контентом.

7. Розробка логіки обробки даних.

7.1. Проєктування структури бази даних.

7.2. Реалізація та оптимізація запитів.

7.3. Аналіз можливості розміщення на сервері.

7.4. Створення механізмів представлення даних.

7.5. Реалізація засобів захисту та контролю доступу.

7.6. Наповнення бази даних інформацією.

7.7. Локальне тестування бази даних.

8. Тестування web-сайту.

8.1. Перевірка коректності дизайну.

8.2. Тестування структури сайту.

8.3. Перевірка логіки функціонування.

8.4. Тестування бази даних.

9. Інсталяція та розгортання.

9.1. Вибір хостинг-платформи.

9.2. Реєстрація та аналіз доменного імені.

9.3. Налаштування взаємодії домену та хостингу.

9.4. Передача продукту замовнику.

10. Супровід та підтримка.

Результати побудови сіткової моделі та розрахунку часових параметрів виконання робіт наведено у табл. 3.1.

Таблиця 3.1 – Результати застосувнаня методу критичного шляху

Номер роботи	Тривалість роботи (діб)	Безпосереднє попередня робота	Безпосереднє наступна робота	Ранній час початку	Ранній час закінч.	Пізній час початку	Пізній час закінч.	Повний резерв часу
1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
1.1	1	-	1.2	0	1	0	1	0
1.2	1	1.1	1.3	1	2	1	2	0
1.3	1	1.2	2.1, 5.1, 7.1	2	3	2	3	0
2.1	2	1.3	2.2	3	5	7	9	4
2.2	2	2.1	3.1	5	7	9	11	4
3.1	0.5	2.2	3.2	7	7.5	11	11.5	4
3.2	0.5	3.1	3.3	7.5	8	11.5	12	4
3.3	1	3.2	3.4	8	9	11	12	3
3.4	0.5	3.3	4	9	9.5	12	12.5	3
4	0.5	3.4	8.1	9.5	10	12.5	13	3
5.1	1	1.3	5.2	3	4	8	9	5
5.2	1	5.1	5.3	4	5	9	10	5
5.3	2	5.2	6	5	7	10	12	5
6	1	5.3	8.2	7	8	12	13	5
7.2	2	7.1	7.3	4	6	4	6	0
7.3	0.5	7.2	7.4	6	6.5	6	6.5	0
7.4	1	7.3	7.5	6.5	7.5	6.5	7.5	0
7.5	2	7.4	7.6	7.5	9.5	7.5	9.5	0
7.6	1	7.5	7.7	9.5	10.5	9.5	10.5	0
7.7	1	7.6	8.4, 8.3, 9.1	10.5	11.5	10.5	11.5	0
8.1	0.5	4	9.4	10.5	10.5	13	13.5	2.5

Продовження таблиці 3.1

1	2	3	4	5	6	7	8	9
8.2	0.5	6	9.4	8.5	10.5	13	13.5	4.5
8.3	0.5	7.7	9.4	11.5	12	13	13.5	1.5
8.4	0.5	7.7	9.4	11.5	12	13	13.5	1.5
9.1	0.5	7.7	9.2	11.5	12	11.5	12	0
9.2	0.5	9.1	9.3	12	12.5	12	12.5	0
9.3	1	9.2	9.4	12.5	13.5	12.5	13.5	0
9.4	0.5	8.1, 8.2, 8.3, 8.4, 9.3	-	13.5	14	13.5	14	0

Аналіз отриманих даних показав, що тривалість критичного шляху становить 14 робочих днів, а майже половина операцій належить до критичних. Така концентрація критичних робіт істотно ускладнює процес управління ризиками, оскільки зменшує можливості маневрування часовими ресурсами [26].

У подібних умовах стандартні підходи до зниження ризиків, зокрема резервування часу, стають малоефективними [31]. Збільшення тривалості робіт, що входять до критичного шляху, неминуче призводить до зростання бюджету та може негативно вплинути на досягнення цілей проєкту [27-30].

З використанням матриць рівнів імовірності та наслідків ризиків формується матриця ідентифікації ризиків, яка наведена у табл. 3.2.

Таблиця 3.2 – Виявлення ризиків

№	Опис ризику	Рівень ймовірності виникнення	Рівень наслідків	Пріоритет
1	**2**	**3**	**4**	**5**
1.	Складна структура сайту	2	3	L
2.	Грубі помилки, виявлені при тестуванні та валідації	3	3	M

Продоження таблиці 3.2

1	2	3	4	5
3.	Конфлікти в графіку	2	4	M
4.	Низька кваліфікація розробників	3	5	H
5.	Складний дизайн	3	2	L
6.	Зміни вимог клієнтом	2	5	M
7.	Нестача матеріальних ресурсів	2	4	M
8.	Помилки в процесі тестування	2	3	L
9.	Некоректна робота серверів, комп’ютерів та програмного забезпечення	2	5	M
10.	Вимоги не відповідають реальності	2	3	L
11.	Затримка при встановленні зв’язку між доменом та хостингом	2	4	M
12.	Звільнення виконавця	2	4	M
13.	Складність структури бази даних	2	4	M
14.	Складність запровадження алгоритму захисту даних	2	4	M
15.	Обрані сервери не підтримують необхідну для розробки СКБД	1	2	L

Після цього виконано процедуру ранжування ризиків та розроблено відповідні плани їх пом’якшення, представлені у табл. 3.3.

Таблиця 3.3 – Стратегії зниження впливу ідентифікованих ризиків

№	Рівень ризику	Критичні етапи	Запропоновані заходи реагування
1	**2**	**3**	**4**
1.	H	Увесь проєкт	Посилення відбору персоналу, проведення навчання та сертифікації
2.	M	3.4, 8.1–8.4	Підвищення якості тестування, додаткові технічні обговорення
3.	M	Увесь проєкт	Оптимізація календарного плану, перерозподіл ресурсів
4.	M	1.1–1.3	Інтенсифікація комунікації з клієнтом, залучення додаткових фахівців
5.	M	7.3, 9.3	Формування фінансового резерву
6.	M	Увесь проєкт	Резервування обладнання та ПЗ
7.	M	9.3	Завчасне налаштування доменно-хостингових зв’язків
8.	M	Увесь проєкт	Планування заміни персоналу
9.	M	7.1–7.7	Ітеративна розробка структури БД
10.	M	7.5	Аналіз альтернативних алгоритмів
11.	L	5.1–5.3	Поступове ускладнення структури
12.	L	2.1–2.2	Оптимізація дизайну
13.	L	3.4, 7.7, 8.1–8.4	Розширення тестових сценаріїв
14.	L	Увесь проєкт	Оцінка доцільності продовження
15.	L	7.3, 7.5	Аналіз альтернативних серверів

За результатами проведеного аналізу ризиків доцільно прийняти управлінське рішення щодо їх активного опрацювання, а не ігнорування чи безумовного прийняття. Як видно з даних, наведених у табл. 3.3, реалізація більшості заходів зниження ризиків пов’язана з додатковими часовими витратами або потребує коригування бюджету проєкту. З урахуванням фактичних умов виконання розробки обґрунтованим є диференційований підхід, за якого ризики з незначним рівнем пріоритету можуть бути прийняті, тоді як ризики середнього та високого рівнів потребують застосування заходів пом’якшення.

Після завершення проєкту інформацію про всі ідентифіковані ризики, у тому числі ті, що не трансформувалися у проблеми, а також ризики, які призвели до матеріальних чи часових втрат, доцільно накопичувати в спеціалізованій базі даних. Використання такої бази знань є особливо ефективним у проєктах web-розробки, оскільки більшість із них мають подібну структуру та передбачають виконання типових процедур.

Це значно спрощує повторну ідентифікацію стандартних ризиків, дозволяючи зосередити основні ресурси на аналізі специфічних загроз і розробці ефективних стратегій їх мінімізації.

3.3. Висновки до третього розділу

1. Проведено тестування та експериментальну перевірку розробленого програмного засобу управління ризиками, у результаті чого підтверджено його відповідність визначеним функціональним вимогам. Реалізований користувацький інтерфейс характеризується інтуїтивністю, логічною структурованістю та зручністю використання, що забезпечує ефективну підтримку процесів формування вимог, ідентифікації ризиків та аналізу їх впливу на програмні проєкти.

2. На основі методу критичного шляху виконано аналіз календарного плану програмного проєкту, у ході якого визначено тривалість критичного шляху та виявлено значну кількість робіт, що мають нульовий резерв часу.

3. З використанням експертних оцінок здійснено ідентифікацію та ранжування ризиків, що можуть виникнути в процесі розробки web-орієнтованого програмного продукту.

4. Розроблено плани пом’якшення ризиків з урахуванням їх пріоритетності та етапів виникнення, які передбачають організаційні, технічні та ресурсні заходи.

5. Обґрунтовано доцільність накопичення інформації про ризики у спеціалізованій базі знань, що дозволяє повторно використовувати результати попередніх проєктів, зменшувати витрати на ідентифікацію типових ризиків та зосереджувати ресурси на аналізі специфічних загроз

4 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

У даному розділі проаналізовано питання охорони праці, де розкрито питання необхідності і способів дотримання вимог охорони праці та техніки безпеки для розробників програмних систем. Окрім цього, у розділі визначено фактори, що впливають на функціональний стан користувачів комп'ютера. Для зменшення негативного впливу таких факторів наведено заходи. Яких необхідно дотримуватися при використанні комп'ютерів та оргтехніки.

4.1. Охорона праці

У кваліфікаційній роботі магістра проведено дослідження методів і програмних засобів управління ризиками у процесі розроблення програмного забезпечення з використанням гнучких (Agile) методологій, а також запропоновано власний підхід до їх реалізації на основі адаптивної моделі SEI. Оскільки проєктування та реалізація програмного засобу управління ризиками виконуються із застосуванням персональних комп'ютерів, засобів розробки програмного забезпечення та мережевих технологій, важливим є дотримання вимог охорони праці та техніки безпеки для розробників програмних систем.

Основним чинним нормативно-правовим документом, який регламентує вимоги безпеки та збереження здоров'я працівників під час роботи з комп'ютерною технікою, є НПАОП 0.00-7.15-18 «Вимоги щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями» [32]. Даний документ встановлює обов'язкові вимоги до організації робочих місць користувачів персональних комп'ютерів, а також до умов експлуатації електронно-обчислювальної техніки.

Зазначений нормативний акт є обов'язковим для виконання роботодавцями та працівниками, діяльність яких пов'язана з використанням електронно-обчислювальних машин, зокрема програмістами, системними аналітиками, тестувальниками та іншими фахівцями у сфері розробки ПЗ [32]. Відповідно до

вимог НПАОП 0.00-7.15-18, комп’ютерна техніка та периферійні пристрої повинні відповідати чинним в Україні стандартам і проходити державну санітарно-епідеміологічну експертизу у встановленому порядку.

Для забезпечення електробезпеки користувачів під час виконання робіт з проєктування та розроблення програмного засобу управління ризиками необхідно, щоб персональні комп’ютери та периферійне обладнання відповідали І класу захисту або були заземлені відповідно до вимог НПАОП 40.1-1.32-01 [32]. При цьому категорично забороняється використання клем функціонального заземлення як елементів захисного заземлення, що прямо регламентовано чинними нормативними документами.

Організація робочих місць розробників програмного забезпечення здійснюється з дотриманням встановлених норм просторового розміщення. Зокрема, відстань між робочим місцем і стіною повинна становити не менше 1 м, а між сусідніми робочими місцями — не менше 1,7 м. Мінімальна площа, що виділяється на одне робоче місце з персональним комп’ютером, повинна бути не меншою за 6,0 м², а об’єм приміщення — не менш як 20 м³ [32].

Під час вибору приміщень для розміщення робочих місць враховано вплив природного освітлення та можливе відбиття світла на екранах моніторів. З метою запобігання осліпленню користувачів комп’ютерів робочі місця розташовуються таким чином, щоб монітори не знаходилися безпосередньо навпроти вікон або поруч із ними [33].

Недостатній рівень освітлення негативно впливає на продуктивність праці та може спричиняти погіршення зору, швидку втому і зниження концентрації уваги. Тому приміщення, у яких здійснюється розроблення програмного забезпечення, обладнані як природним, так і штучним освітленням. Розміщення робочих місць із персональними комп’ютерами у підвальних приміщеннях не допускається [32].

У випадках виконання робіт, що потребують високого рівня розумової концентрації, зокрема під час аналізу ризиків, моделювання Agile-процесів та реалізації алгоритмів обробки даних, робочі місця ізольовуються між собою перегородками висотою не менше 1,6 м [32]. Підлогове покриття у приміщеннях

повинно бути рівним, нековзким та придатним для регулярного вологого прибирання.

Штучне освітлення в робочих приміщеннях реалізоване у вигляді комбінованої системи з використанням люмінесцентних світильників загального освітлення, розташованих рівномірно над робочими зонами. Освітленість на робочих місцях із ПК повинна знаходитися у межах 300–500 лк, що відповідає вимогам чинних нормативних документів [32].

Для зменшення впливу прямих світлових потоків на екрани моніторів світильники розташовуються з бічним зміщенням відносно робочих місць та паралельно до світлових отворів. Вікна обладнані світлорозсіюючими шторами або жалюзі з регульованими ламелями та коефіцієнтом відбивання не менше 0,7 [33].

З метою запобігання нещасним випадкам і забезпечення належного рівня охорони праці в установі розробляються та впроваджуються інструкції з охорони праці та техніки безпеки при роботі з комп’ютерною технікою. Дія таких інструкцій поширюється на всі структурні підрозділи організації [33].

До виконання робіт з розроблення програмного забезпечення допускаються лише працівники, які пройшли відповідне навчання, медичний огляд, вступний інструктаж з охорони праці, інструктаж на робочому місці та інструктаж з пожежної безпеки [33].

З урахуванням ергономічних вимог під час організації робочих місць враховано такі фактори:

– достатній простір для розміщення користувача;
– зручність огляду елементів робочого місця.

Таким чином, у процесі дослідження та розроблення методу і програмного засобу управління ризиками у гнучких методологіях виконано аналіз чинних нормативних вимог з охорони праці та техніки безпеки. Дотримання зазначених вимог дозволило створити безпечні, комфортні та ефективні умови праці для розробників ПЗ, що є важливим чинником підвищення якості та надійності програмних продуктів.

4.2. Фактори, що впливають на функціональний стан користувачів комп’ютера

Робота відеотерміналів включає різні завдання, які об'єднуються такими загальними чинниками, як те, що робота проводиться в сидячому положенні і вимагає уважного, неперервного та іноді тривалого спостереження [34].

Виділяють три групи основних завдань, які розв'язуються на відеотерміналах:

– контроль і спостереження;

– діалог;

– збір інформації.

Ці завдання розрізняються по тривалості використання дисплея і по ступеню уваги, якого вони вимагають. Важливим питанням є режим праці і відпочинку при роботі з відеотерміналами. Виділяють 7 умов для того, щоб діяльність на робочому місці, оснащеному дисплеєм, здійснювалася без скарг і без втоми [34].

Правильне облаштування робочого столу:

– при фіксованій висоті – оптимальна висота - 720мм;

– повинен забезпечуватися необхідний простір для рук по висоті, ширині і глибині;

– в області сидіння не повинно бути шухляд.

Правильне встановлення робочого стільця:

– висота повинна регулюватися;

– конструкція повинна бути такою, що обертається;

– правильна висота сидіння: площа сидіння на 30мм нижче, ніж підколінна западина.

Правильне розташування приладів: необхідно так установити яскравість знаків і яскравість фону дисплея, щоб не було великої відмінності в порівнянні з яскравістю навколишнього оточення, але щоб знаки чітко пізнавалися на відстані читання. Не допускати, згідно [35]:

– дуже велику яскравість (викликає мерехтіння);

– дуже слабку яскравість (сильне навантаження на очі);

– дуже чорну фонову яскравість дисплея (сильне навантаження а очі).

Правильне виконання робіт:

– положення тулуба пряме, ненапружене;

– положення голови пряме, вільне, зручне;

– положення рук - зігнуті трохи більше, ніж під прямим кутом;

– положення ніг - зігнуті трохи більше, ніж під прямим кутом;

– правильна відстань для зору, клавіатура і дисплей –приблизно на однаковій відстані для зору: при постійній роботі - близько 500мм, при випадковій роботі - до 700мм.

Правильне освітлення:

– освітлення по можливості із сторони, зліва;

– по можливості - рівномірне освітлення всього робочого простору;

– прилади по можливості встановлювати в місцях, віддалених від вікон;

– вибирати непряме освітлення приміщення або вкривати корпуси світильників;

– світло, що поступає через вікна, пом'якшувати за допомогою штор;

– організувати робоче місце, щоб напрям погляду йшов по можливості паралельно фронту вікон.

Правильне застосування допоміжних засобів: підлокітники використовувати, якщо клавіатура вища 15мм [35].

Правильний метод роботи:

– передбачати по можливості зміну завдань і навантажень;

– дотримувати перерви в роботі: 5 хвилин через 1 годину роботи біля дисплея або 10 хвилин після 2-х годин роботи біля дисплея.

При створенні сприятливих умов для підвищення продуктивності і зменшення напруги значну роль грають чинники, що характеризують стан навколишнього середовища: мікроклімат приміщення, рівень шуму і освітлення. рекомендована величина відносної вологості - 65-70%. робоче місце повинне добре вентилюватися. В даний час з погляду шумового навантаження досягнутий значний

прогрес. Рівень шуму в залі (приблизно 40дб) не перевищує рівень КБ, незалежно від кількості використовуваної апаратури. По останніх дослідженнях - робота за відеотерміналом не представляє небезпеки з погляду рентгенівського випромінювання [35].

Ергономічна організація робочого місця користувача ЕОМ повинна враховувати як специфіку діяльності, що виконується, так забезпечувати комфортні умови перебування людини.

Тому до основних ергономічних завдань щодо організації робочого місця слід віднести [35]:

– забезпечення просторових параметрів робочого місця, які відповідають антропометричним характеристикам користувача;

– раціональне розташування елементів робочого місця відносно користувача на підставі поглибленого кількісного та якісного аналізу діяльності, яка виконується;

– оптимізацію умов робочого середовища.

В ході організації робочих місць на кожну ЕОМ повинна бути виділена площа, яка складає не менш, ніж 6 $м^2$, та об'єм, який становить не менш, ніж 24 $м^3$. Причому, зона, де розташовується робочий стіл, сервер або робоча станція, принтер, екран для графопроектора, повинна займати відповідно 6 - 8 $м^2$. Висота приміщення повинна бути не менш, ніж 4 м [35].

Робоче місце користувача ПК повинно бути обладнане одномісним столом та напівм'яким стільцем, висоту сидіння яких можна змінювати. Довжина стола повинна бути не менше 70 см, ширина – забезпечувати місце перед клавіатурою (не менше, ніж 40 см) для розташування зошита або іншого приладдя. Поверхня стола повинна мати кут нахилу у межах 12-150, лише іноді припустимою є її розташування у горизонтальній площині.

Слід відмітити, що оптимальна відстань від очей до площини екрана монітора, повинна складати 60–70 см, припустима – не менше 50 см. Розглядати інформацію на екрані з відстані менш, ніж 50 см не рекомендується.

ВИСНОВКИ

У кваліфікаційній роботі магістра одержано наступні наукові і практичні результати.

1. На основі аналізу гнучких методологій розробки програмного забезпечення встановлено, що Scrum і DSDM забезпечують високу адаптивність процесів, проте потребують доповнення формалізованими механізмами управління ризиками. Це обґрунтовує доцільність інтеграції моделей ризик-менеджменту в Agile-процеси з метою підвищення передбачуваності результатів розробки.

2. Досліджено модель SEI як інструмент формалізованої ідентифікації та класифікації ризиків програмних проєктів. Визначено основні групи джерел ризиків (технічні, вартісні, планові та управлінські), що дозволяє комплексно охопити фактори, які впливають на якість, строки та вартість реалізації програмного продукту.

3. Обґрунтовано доцільність використання UML як засобу структурованого опису програмного проєкту з метою підтримки процесу ідентифікації ризиків на ранніх етапах життєвого циклу. Встановлено, що застосування UML-діаграм дає змогу ідентифікувати більшість атрибутів технічних ризиків відповідно до моделі SEI.

4. За результатами аналізу визначено, що поєднання Agile-методологій, моделі SEI та UML-моделювання створює методологічну основу для розробки адаптованої технології управління, оцінювання та прогнозування ризиків програмного забезпечення, що й визначає напрям подальших досліджень у наступних розділах роботи.

5. Проведено аналіз предметної області та ролей учасників процесу розробки, у результаті якого визначено функціональні обов’язки системного аналітика, експерта з ризиків і менеджера проєкту в контексті Agile-процесів. Побудовані UML-діаграми прецедентів формалізують сценарії взаємодії

користувачів із системою та створюють основу для автоматизації процесів ідентифікації, аналізу та моніторингу ризиків.

6. Запропоновано багаторівневу архітектуру програмного засобу управління ризиками, яка включає рівні представлення, прикладної логіки, доменної моделі, доступу до даних і бази даних. Такий підхід забезпечує розділення відповідальностей, слабке зв’язування компонентів та можливість подальшого масштабування або інтеграції системи з іншими програмними рішеннями.

7. Розроблено діаграми пакетів і класів, що відображають логічну структуру програмного засобу та формалізують ключові сутності предметної області: проєкти, спринти, задачі, функціональність, метрики якості та повну модель ризиків SEI. Це дало змогу чітко визначити взаємозв’язки між елементами Agile-процесу та ризиками, а також створити основу для реалізації алгоритмів оцінювання і прогнозування ризиків.

8. Спроєктовано структуру реляційної бази даних у вигляді ER-діаграми, яка узгоджується з доменною моделлю та забезпечує нормалізоване зберігання даних про проєкти, планування робіт, вимоги, метрики та ризики. Запропонована модель БД дозволяє накопичувати історію змін ризиків, аналізувати їх динаміку та використовувати ці дані для підтримки прийняття управлінських рішень у подальших проєктах.

9. Сформовано цілісну концептуальну та архітектурну основу програмного засобу, що реалізує метод управління ризиками на основі моделі SEI в Agile-середовищі. Це створює передумови для подальшої реалізації програмного продукту, його практичної апробації та розширення функціональності шляхом автоматизованого аналізу, моніторингу й прогнозування ризиків програмного забезпечення.

10. Обґрунтовано доцільність накопичення інформації про ризики у спеціалізованій базі знань, що дозволяє повторно використовувати результати попередніх проєктів, зменшувати витрати на ідентифікацію типових ризиків та зосереджувати ресурси на аналізі специфічних загроз

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Sommerville I. Software Engineering. 10th ed. Boston: Pearson Education. 2016. 816 p.

2. Boehm B., Turner R. Management Challenges to Implementing Agile Processes in Traditional Development Organizations. IEEE Software. 2015. Vol. 32, No. 5. PP. 15–18.

3. Schwaber K., Sutherland J. The Scrum Guide. Scrum.org. 2020. 13 p.

4. Kerzner H. Project Management: A Systems Approach to Planning, Scheduling, and Controlling. 12th ed. Hoboken: Wiley. 2017. 832 p.

5. Leffingwell D. SAFe® 4.0 Reference Guide: Scaled Agile Framework for Lean Enterprises. Boston: Addison-Wesley. 2016. 560 p.

6. Bannerman P. Risk management in software projects: A reassessment. Journal of Systems and Software. 2015. Vol. 100. PP. 1–15.

7. Wallace L., Keil M., Rai A. Understanding software project risk: A cluster analysis. Information & Management. 2016. Vol. 53, No. 3. PP. 341–354.

8. Verner J. M., Brereton O. P., Kitchenham B. A. Risks and risk mitigation in global software development: A tertiary study. Information and Software Technology. 2015. Vol. 56. PP. 54–78.

9. Pichler R. Agile Product Management with Scrum. Boston: Addison-Wesley. 2016. 192 p.

10. Forsgren N., Humble J., Kim G. Accelerate: The Science of Lean Software and DevOps. Portland: IT Revolution Press. 2018. 288 p.

11. Chrissis M. B., Konrad M., Shrum S. CMMI for Development, Version 2.0. Pittsburgh: Software Engineering Institute. 2018. 846 p.

12. SEI. CMMI V2.0 Model Overview. Carnegie Mellon University. 2018. PP. 1–42.

13. ISO/IEC 12207:2017 Systems and Software Engineering — Software Life Cycle Processes.

14. ISO/IEC 25010:2011 / Amd.1:2015 Systems and Software Quality Models.

15. ISO 31000:2018 Risk Management — Guidelines.

16. IEEE Std 1540-2016 IEEE Standard for Software Life Cycle Processes — Risk Management.

17. Pastukh O., Yatsyshyn V. Development of software for neuromarketing based on artificial intelligence and data science using high-performance computing and parallel programming technologies. Scientific Journal of TNTU (Tern.). vol 113. no 1. 2024. pp. 143–149.

18. Stefanyshyn V., Stefanyshyn I., Pastukh O., Yatsyshyn V., Yakymenko I. Accuracy of software and hardware of computer systems for human-machine interaction CEUR Workshop Proceedings, Volume 3842, 1st International Workshop on Bioinformatics and Applied Information Technologies, BAIT 2024 Zboriv 2 October 2024 through 4 October 2024 Code 204273 pp. 178–183.

19. Pastukh O., Yatsyshyn V., Kukharska V., Palamar A., Kulikov S. Method and tool of detecting software architecture patterns in the process of computer systems development. CEUR Workshop Proceedings,Volume 3896, 2024 4th International Workshop on Information Technologies: Theoretical and Applied Problems, ITTAP 2024 Ternopil 23 October 2024 through 25 October 2024 Code 206051. Ukraine and Opole, Poland. Pages 12 - 24.

20. O. Pastukh, V. Yatsyshyn, A. Lutskiv, V. Tsymbalistyy, N. Martsenko. A Risks management method based on the quality requirements communication method in agile approaches. Information technologies: theoretical and applied problems, 2022. P. 1-10

21. Pastukh O, Yatsyshyn V., Palamar A., Zharovskyi R. Technology of relational database management systems performance evaluation during computer systems design. Scientific Journal of TNTU.Tern.: TNTU. 2023. Vol 109. No 1. P. 54–65.

22. Pastukh O., Yatsyshyn V., Zharovskyi R., Shabliy N. Software tool for productivity metrics measure of relational database management system. Mathematical Modeling. No 1 (48). 2023. P. 7-17

23. Пастух О.А., Новицька Х.О. Моделювання процесів управління ризиками в життєвому циклі програмного забезпечення із застосуванням UML та

адаптованої SEI-моделі. Матеріали XIV міжнародної науково - технічної конференції молодих учених і студентів «Актуальні задачі сучасних технологій» (11-12 грудня 2025 р.) Тернопільського національного технічного університету імені Івана Пулюя. Тернопіль: ТНТУ. 2025. С. 315-318.

24. Пастух О.А., Новицька Х.О. Архітектура програмної системи управління ризиками на основі адаптованої моделі SEI. Матеріали XIII науково-технічної конференції Тернопільського національного технічного університету імені Івана Пулюя «Інформаційні моделі, системи та технології» (17-18 грудня 2025 року). Тернопіль: ТНТУ. 2025. С. 186.

25. Dingsoyr T., Moe N. B. Research Challenges in Large-Scale Agile Software Development. ACM SIGSOFT Software Engineering Notes. 2015. Vol. 40, No. 2. PP. 38–39.

26. Šmite D., Moe N. B. Risks in distributed agile development. IEEE Software. 2016. Vol. 33, No. 2. PP. 62–68.

27. Ramesh B., Cao L., Baskerville R. Agile requirements engineering practices and challenges. Information Systems Journal. 2016. Vol. 26, No. 4. PP. 387–414.

28. Kitchenham B., Charters S. Guidelines for Performing Systematic Literature Reviews in Software Engineering. EBSE Technical Report. 2015. PP. 1–65.

29. Stellman A., Greene J. Learning Agile: Understanding Scrum, XP, Lean, and Kanban. Sebastopol: O'Reilly Media. 2015. 448 p.

30. Rubin K. S. Essential Scrum: A Practical Guide to the Most Popular Agile Process. Boston: Addison-Wesley. 2016. 512 p.

31. Boehm B., Lane J. Using the Incremental Commitment Model to integrate system engineering and software engineering. CrossTalk. 2015. Vol. 28, No. 5. PP. 4–10.

32. Бедрій Я. Основи охорони праці користувачів персональних комп'ютерів: навчальний посібник для студентів ВНЗ та інженерів-практиків. Навчальна книга-Богдан. 2014. 144 с.

33. НПАОП 0.00-7.15-18 «Вимоги щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями». Київ. 2018.

34. Стручок В.С. Методичний посібник для здобувачів освітнього ступеня «магістр» всіх спеціальностей денної та заочної (дистанційної) форм навчання «Безпека в надзвичайних ситуаціях» / Тернопіль: ФОП Паляниця В. А.156 с.

35. Стручок В.С. Навчальний посібник «Техноекологія та цивільна безпека. Частина «цивільна безпека»» / Тернопіль: ФОП Паляниця В. А. 156 с.

ДОДАТКИ

ДОДАТОК А

Апробація результатів роботи

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Тернопільський національний технічний університет імені Івана Пулюя
Маріборский університет (Словенія)
Технічний університет у Кошице (Словаччина)
Вільнюський технічний університет ім. Гедимінаса (Литва)
Краківський економічний університет (Польща)
Вроцлавський економічний університет (Польща)
Університет «Опольська Політехніка» (Польща)
Національний університет «Полтавська політехніка імені Юрія Кондратюка»
Вінницький національний аграрний університет
Львівський національний університет ім. І. Франка
Головне управління Пенсійного фонду в Тернопільській області
Наукове товариство ім. Шевченка
Тернопільський обласний комунальний інститут післядипломної педагогічної освіти
Сумський державний педагогічний університет
Запорізький національний університет

АКТУАЛЬНІ ЗАДАЧІ СУЧАСНИХ ТЕХНОЛОГІЙ

Збірник

тез доповідей

XIV Міжнародної науково-технічної конференції молодих учених та студентів

11-12 грудня 2025 року

УКРАЇНА
ТЕРНОПІЛЬ – 2025

використовується в архівах і репозиторіях для підтримки стійких процесів цифрового збереження.

Для уніфікації обміну інформацією між установами застосовуються міжнародні стандарти метаданих, такі як Dublin Core, LIDO (Lightweight Information Describing Objects) та CIDOC CRM. Вони дають змогу забезпечити взаємодію між різними інформаційними системами наприклад, Omeka, CollectiveAccess, Axiell Collections без втрати логічної структури даних [4]. Відповідно до рекомендацій ЮНЕСКО, побудова таких відкритих інфраструктур повинна спиратися на принципи інтероперабельності, відкритих форматів і сталості даних

Отже, інформаційні моделі та метадані є фундаментом цифрового збереження музейних фондів. Вони забезпечують структурованість, автентичність, довготривалу стабільність і відкритість даних, а також сприяють інтеграції українських музеїв у міжнародний цифровий простір. Метадані перестають бути лише технічним атрибутом, вони стають інтелектуальною мовою комунікації культурної спадщини у глобальному цифровому середовищі.

Література

1. Doerr M. The CIDOC Conceptual Reference Model: An Ontological Approach to Semantic Interoperability of Metadata. AI Magazine, 2003. URL: https://cidoc-crm.org/
2. Hyvönen E. Publishing and Using Cultural Heritage Linked Data on the Semantic Web. Morgan & Claypool, 2012. URL: https://seco.cs.aalto.fi/publications/2012/hyvonen-linkeddata/
3. PREMIS Editorial Committee. Data Dictionary for Preservation Metadata. Library of Congress, 2022. URL: https://www.loc.gov/standards/premis/
4. UNESCO. Guidelines for Digital Heritage Information Systems. Paris, 2020. URL: https://unesdoc.unesco.org/ark:/48223/pf0000375748

УДК 004.42

О.А. Пастух докт. техн. наук, професор, Х.О. Новицька
Тернопільський національний технічний університет імені Івана Пулюя

МОДЕЛЮВАННЯ ПРОЦЕСІВ УПРАВЛІННЯ РИЗИКАМИ В ЖИТТЄВОМУ ЦИКЛІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ІЗ ЗАСТОСУВАННЯМ UML ТА АДАПТОВАНОЇ SEI-МОДЕЛІ

O.A. Pastukh Dr., Prof., Kh.O. Novytska

MODELING RISK MANAGEMENT PROCESSES IN THE SOFTWARE LIFE CYCLE USING UML AND AN ADAPTED SEI-BASED APPROACH

Комплексне вивчення предметної області та ключових процесів життєвого циклу програмного забезпечення, зокрема тих, що пов'язані з управлінням ризиками, є важливою передумовою для побудови програмної системи, спрямованої на підвищення результативності розробки програмних продуктів. Аналіз зазначених процесів дозволяє визначити основні вимоги до системи, виокремити критичні аспекти та сформувати концептуальну модель, що стане основою для подальшого проектування.

Одним із найзручніших та загальноприйнятих засобів об'єктно-орієнтованого моделювання є UML. Дана мова забезпечує широкий спектр графічних нотацій для опису структури та поведінки системи. У межах аналізу предметної області особливу

цінність становлять такі діаграми, як діаграма варіантів використання, діаграма класів, діаграма компонентів та діаграма видів діяльності.

Реалізація підходу управління ризиками в процесі проектування програмних рішень, який інтегрується у методології Agile та спирається на адаптовану модель SEI, потребує участі кількох ролей як: системний аналітик, експерт з ризиків та менеджер проєкту.

На рис. 1 відображено діаграму прецедентів, що демонструє основні функції експерта в межах Agile-процесів.

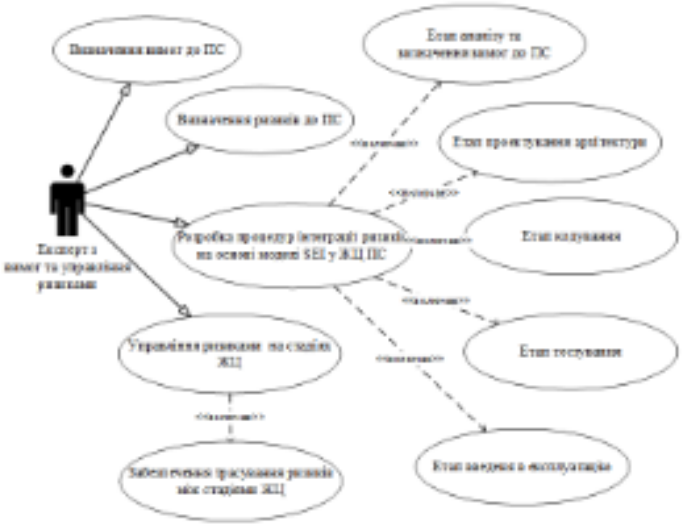

Рис. 1. Діаграма прецедентів «Експерт»

Менеджер проєкту, за умов використання гнучких методологій, виконує ключову роль в організації робочих процесів. Він контролює виконання етапів проекту, слідкує за відповідністю результатів вимогам, реагує на ризики та зміни. На рис. 2 подано діаграму прецедентів, що відображає додаткові функції менеджера, необхідні в умовах підвищених вимог до управління ризиками.

Рис. 2. Діаграма прецедентів «Менеджер проєкту»

Системний аналітик, інтегруючи запропоновану модель управління ризиками, повинен забезпечити трансформацію потреб користувачів у формальні вимоги, враховуючи їх якісні характеристики, а також забезпечити їх узгодження з визначеними ризиками. На рис. 3 зображено основні процеси, автоматизація яких є необхідною для ефективної роботи аналітика.

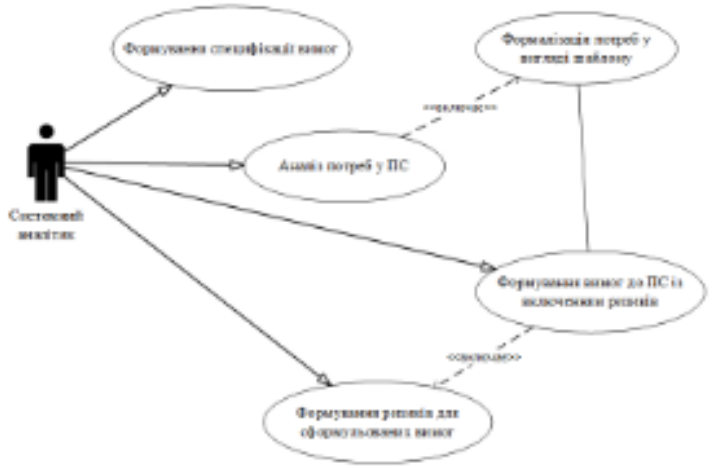

Рис. 3. Діаграма прецедентів «Системний аналітик»

До компетенції системного аналітика входить формування узгодженого набору вимог до програмного продукту з одночасним врахуванням ризиків, які були визначені експертом. Ключовим завданням є забезпечення їх трасування – тобто простежуваності вимог і ризиків між різними етапами життєвого циклу системи.

Отже, розроблені діаграми прецедентів описують взаємодію всіх учасників процесу розробки, забезпечують системний підхід до управління ризиками відповідно до SEI-моделі та створюють основу для подальшого проектування архітектури програмного засобу підтримки процесів управління ризиками.

УДК 004.72
С. Орлов, аспірант, С. Марценко, к.т.н., доц.
(Тернопільський національний технічний університет ім. І. Пулюя, Україна)

ДОСЛІДЖЕННЯ ІТ ЛОГІСТИЧНИХ МОДЕЛЕЙ

S. Orlov, PhD student, S. Martsenko, Ph.D., Assoc. Prof

RESEARCH OF IT LOGISTICS MODELS

Активний розвиток електронної комерції та глобальна цифровізація усіх сфер життя змінюють вимоги до ІТ логістичних моделей і їх використання. Сюди можна віднести концептуальні та математичні моделі, що впливають на підтримку, оптимізацію та автоматизацію логістичних процесів [1].

Структурні моделі відповідають за відображення загальної будови логістичних систем та описують ланцюг постачання, взаємодію складів, транспортних вузлів і виробничих потужностей. Прикладом є модель SCOR (Supply Chain Operations Reference), яка використовується для аналізу та вдосконалення ланцюгів постачання.

Процесні моделі в основному описують бізнес-процеси, такі як закупівля, складування, транспортування та управління замовленнями. Основною ціллю таких

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

М А Т Е Р І А Л И

XIII НАУКОВО-ТЕХНІЧНОЇ КОНФЕРЕНЦІЇ

«ІНФОРМАЦІЙНІ МОДЕЛІ, СИСТЕМИ ТА ТЕХНОЛОГІЇ»

17-18 грудня 2025 року

ТЕРНОПІЛЬ
2025

УДК 004.4
О. Пастух, д. т. н., проф.;Х. Новицька
(Тернопільський національний технічний університет імені Івана Пулюя, Україна)

АРХІТЕКТУРА ПРОГРАМНОЇ СИСТЕМИ УПРАВЛІННЯ РИЗИКАМИ НА ОСНОВІ АДАПТОВАНОЇ МОДЕЛІ SEI

UDC 004.4
O. Pastukh, Dr., Prof.; K. Novytska

ARCHITECTURE OF A RISK MANAGEMENT SOFTWARE SYSTEM BASED ON AN ADAPTED SEI MODEL

Архітектуру системи управління ризиками ПЗ у гнучких методологіях на основі моделі SEI запропоновано представити на вигляді багатошарової структури (рис. 1).

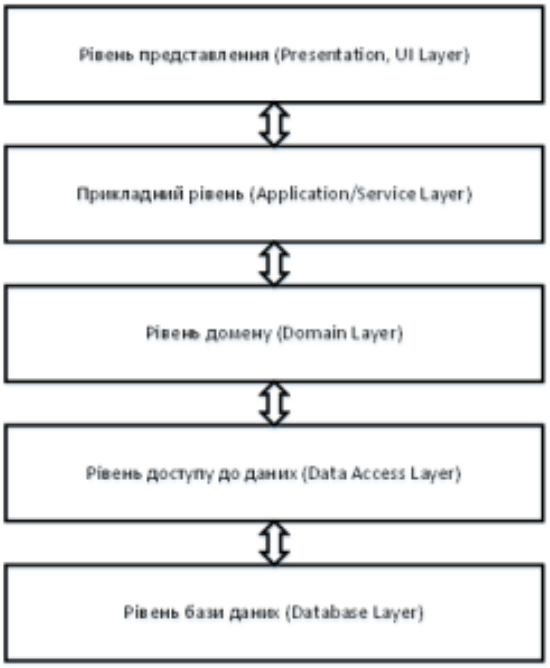

Рисунок 1. Багаторівнева архітектура програмної системи управління ризиками на основі адаптованої моделі SEI

Основними рівнями архітектури є:

1. Рівень представлення – містить форми користувацького інтерфейсу.
2. Прикладний рівень – реалізує фасади і сервіси, що реалізують сценарії використання.
3. Доменний рівень – модель предметної області: проєкти, спринти, задачі, функціональність, метрики, ризики SEI.
4. Рівень доступу до даних – репозиторії та засоби роботи з СКБД.
5. Рівень бази даних – реалізація у вигляді реляційної БД.

Комунікація між шарами виконується зверху вниз: UI викликає сервіси, сервіси працюють з доменними об'єктами через репозиторії, а репозиторії взаємодіють із БД. Зворотні залежності (знизу вгору) реалізуються тільки через передавання результатів викликів, що забезпечує слабке зв'язування.

ДОДАТОК Б

Скрипт генерації бази даних

```
USE [master]
GO
/****** Object:  Database [RISK_MANAGEMENT]  Script Date: 17.12.2025 10:46:04 ******/
CREATE DATABASE [RISK_MANAGEMENT]
 CONTAINMENT = NONE
 ON  PRIMARY
( NAME = N'RISK_MANAGEMENT', FILENAME = N'C:\Program Files\Microsoft SQL Server\MSSQL14.SQLEXPRESS\MSSQL\DATA\RISK_MANAGEMENT.mdf' , SIZE = 8192KB , MAXSIZE = UNLIMITED, FILEGROWTH = 65536KB )
 LOG ON
( NAME = N'RISK_MANAGEMENT_log', FILENAME = N'C:\Program Files\Microsoft SQL Server\MSSQL14.SQLEXPRESS\MSSQL\DATA\RISK_MANAGEMENT_log.ldf' , SIZE = 8192KB , MAXSIZE = 2048GB , FILEGROWTH = 65536KB )
GO
ALTER DATABASE [RISK_MANAGEMENT] SET COMPATIBILITY_LEVEL = 140
GO
IF (1 = FULLTEXTSERVICEPROPERTY('IsFullTextInstalled'))
begin
EXEC [RISK_MANAGEMENT].[dbo].[sp_fulltext_database] @action = 'enable'
end
GO
ALTER DATABASE [RISK_MANAGEMENT] SET ANSI_NULL_DEFAULT OFF
GO
ALTER DATABASE [RISK_MANAGEMENT] SET ANSI_NULLS OFF
GO
ALTER DATABASE [RISK_MANAGEMENT] SET ANSI_PADDING OFF
GO
ALTER DATABASE [RISK_MANAGEMENT] SET ANSI_WARNINGS OFF
```

```
GO
ALTER DATABASE [RISK_MANAGEMENT] SET ARITHABORT OFF
GO
ALTER DATABASE [RISK_MANAGEMENT] SET AUTO_CLOSE ON
GO
ALTER DATABASE [RISK_MANAGEMENT] SET AUTO_SHRINK OFF
GO
ALTER DATABASE [RISK_MANAGEMENT] SET AUTO_UPDATE_STATISTICS ON
GO
ALTER DATABASE [RISK_MANAGEMENT] SET CURSOR_CLOSE_ON_COMMIT OFF
GO
ALTER DATABASE [RISK_MANAGEMENT] SET CURSOR_DEFAULT  GLOBAL
GO
ALTER DATABASE [RISK_MANAGEMENT] SET CONCAT_NULL_YIELDS_NULL OFF
GO
ALTER DATABASE [RISK_MANAGEMENT] SET NUMERIC_ROUNDABORT OFF
GO
ALTER DATABASE [RISK_MANAGEMENT] SET QUOTED_IDENTIFIER OFF
GO
ALTER DATABASE [RISK_MANAGEMENT] SET RECURSIVE_TRIGGERS OFF
GO
ALTER DATABASE [RISK_MANAGEMENT] SET  ENABLE_BROKER
GO
ALTER          DATABASE          [RISK_MANAGEMENT]          SET
AUTO_UPDATE_STATISTICS_ASYNC OFF
GO
ALTER          DATABASE          [RISK_MANAGEMENT]          SET
DATE_CORRELATION_OPTIMIZATION OFF
GO
ALTER DATABASE [RISK_MANAGEMENT] SET TRUSTWORTHY OFF
GO
ALTER DATABASE [RISK_MANAGEMENT] SET ALLOW_SNAPSHOT_ISOLATION OFF
GO
ALTER DATABASE [RISK_MANAGEMENT] SET PARAMETERIZATION SIMPLE
GO
ALTER DATABASE [RISK_MANAGEMENT] SET READ_COMMITTED_SNAPSHOT OFF
```

```
GO
ALTER DATABASE [RISK_MANAGEMENT] SET HONOR_BROKER_PRIORITY OFF
GO
ALTER DATABASE [RISK_MANAGEMENT] SET RECOVERY SIMPLE
GO
ALTER DATABASE [RISK_MANAGEMENT] SET  MULTI_USER
GO
ALTER DATABASE [RISK_MANAGEMENT] SET PAGE_VERIFY CHECKSUM
GO
ALTER DATABASE [RISK_MANAGEMENT] SET DB_CHAINING OFF
GO
ALTER DATABASE [RISK_MANAGEMENT] SET FILESTREAM( NON_TRANSACTED_ACCESS = OFF )
GO
ALTER DATABASE [RISK_MANAGEMENT] SET TARGET_RECOVERY_TIME = 60 SECONDS
GO
ALTER DATABASE [RISK_MANAGEMENT] SET DELAYED_DURABILITY = DISABLED
GO
ALTER DATABASE [RISK_MANAGEMENT] SET QUERY_STORE = OFF
GO
USE [RISK_MANAGEMENT]
GO
/****** Object:  Table [dbo].[Feature]    Script Date: 17.12.2025 10:46:04 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[Feature](
	[ID_Feature] [int] IDENTITY(1,1) NOT NULL,
	[ID_Task] [int] NULL,
	[FeatureTitle] [nvarchar](150) NULL,
	[IntegralQualityValue] [int] NULL,
	[Description] [nvarchar](max) NULL,
```

```
	PRIMARY KEY CLUSTERED
	(
		[ID_Feature] ASC
	)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON
[PRIMARY]
	) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
	GO
	/****** Object: Table [dbo].[Feature_Metric] Script Date:
17.12.2025 10:46:04 ******/
	SET ANSI_NULLS ON
	GO
	SET QUOTED_IDENTIFIER ON
	GO
	CREATE TABLE [dbo].[Feature_Metric](
		[ID_Feature] [int] NOT NULL,
		[ID_Metric] [int] NOT NULL,
		[CurrentMetricValue] [float] NULL,
	PRIMARY KEY CLUSTERED
	(
		[ID_Feature] ASC,
		[ID_Metric] ASC
	)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON
[PRIMARY]
	) ON [PRIMARY]
	GO
	/****** Object: Table [dbo].[Metric] Script Date: 17.12.2025
10:46:04 ******/
	SET ANSI_NULLS ON
	GO
	SET QUOTED_IDENTIFIER ON
	GO
	CREATE TABLE [dbo].[Metric](
		[ID_Metric] [int] IDENTITY(1,1) NOT NULL,
		[MetricTitle] [nvarchar](150) NULL,
```

```
	[MetricValue] [float] NULL,
	[Description] [nvarchar](max) NULL,
PRIMARY KEY CLUSTERED
(
	[ID_Metric] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON
[PRIMARY]
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO
/****** Object:  Table [dbo].[Project]    Script Date: 17.12.2025
10:46:04 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[Project](
	[ID_Project] [int] IDENTITY(1,1) NOT NULL,
	[ProjectTitle] [nvarchar](150) NULL,
	[Description] [nvarchar](max) NULL,
PRIMARY KEY CLUSTERED
(
	[ID_Project] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON
[PRIMARY]
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO
/****** Object:  Table [dbo].[Risk]    Script Date: 17.12.2025
10:46:04 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[Risk](
	[ID_Risk] [int] IDENTITY(1,1) NOT NULL,
```

```
	[ID_RiskType] [int] NULL,
	[RiskTitle] [nvarchar](150) NULL,
	[Description] [nvarchar](max) NULL,
	[ID_SourceRisk] [int] NULL,
PRIMARY KEY CLUSTERED
(
	[ID_Risk] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO
/****** Object:  Table [dbo].[Risk_Metric]    Script Date: 17.12.2025 10:46:04 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[Risk_Metric](
	[ID_Risk] [int] NOT NULL,
	[ID_Metric] [int] NOT NULL,
	[Value] [float] NULL,
	[Date] [datetime] NULL,
PRIMARY KEY CLUSTERED
(
	[ID_Risk] ASC,
	[ID_Metric] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
/****** Object:  Table [dbo].[Risk_Type]    Script Date: 17.12.2025 10:46:04 ******/
SET ANSI_NULLS ON
GO
```

```
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[Risk_Type](
	[ID_RiskType] [int] IDENTITY(1,1) NOT NULL,
	[RiskTypeValue] [int] NULL,
	[Description] [nvarchar](max) NULL,
PRIMARY KEY CLUSTERED
(
	[ID_RiskType] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON
[PRIMARY]
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO
/****** Object: Table [dbo].[RiskContext] Script Date:
17.12.2025 10:46:04 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[RiskContext](
	[ID_RiskContext] [int] IDENTITY(1,1) NOT NULL,
	[ContextLevel] [varchar](20) NOT NULL,
	[ID_Project] [int] NULL,
	[ID_Sprint] [int] NULL,
	[ID_Task] [int] NULL,
	[ID_Feature] [int] NULL,
	[ID_Risk] [int] NULL,
	[Comment] [nvarchar](500) NULL,
	[CreatedAt] [datetime] NOT NULL,
PRIMARY KEY CLUSTERED
(
	[ID_RiskContext] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON
[PRIMARY]
```

```
) ON [PRIMARY]
GO
/****** Object:  Table [dbo].[SourceRisk]  Script Date: 17.12.2025 10:46:04 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[SourceRisk](
	[ID_SourceRisk] [int] IDENTITY(1,1) NOT NULL,
	[Code] [int] NULL,
	[Description] [nvarchar](max) NULL,
PRIMARY KEY CLUSTERED
(
	[ID_SourceRisk] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO
/****** Object:  Table [dbo].[Sprint]  Script Date: 17.12.2025 10:46:04 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[Sprint](
	[ID_Sprint] [int] IDENTITY(1,1) NOT NULL,
	[ID_Project] [int] NULL,
	[SprintTitle] [nvarchar](150) NULL,
	[Description] [nvarchar](max) NULL,
PRIMARY KEY CLUSTERED
(
	[ID_Sprint] ASC
```

```
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO
/****** Object: Table [dbo].[Sprint_Task] Script Date: 17.12.2025 10:46:04 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[Sprint_Task](
	[ID_Sprint] [int] NOT NULL,
	[ID_Task] [int] NOT NULL,
PRIMARY KEY CLUSTERED
(
	[ID_Sprint] ASC,
	[ID_Task] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
/****** Object: Table [dbo].[Task] Script Date: 17.12.2025 10:46:04 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[Task](
	[ID_Task] [int] IDENTITY(1,1) NOT NULL,
	[ID_Project] [int] NULL,
	[TaskTitle] [nvarchar](150) NULL,
	[Description] [nvarchar](max) NULL,
PRIMARY KEY CLUSTERED
(
```

```
        [ID_Task] ASC
    )WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON
[PRIMARY]
    ) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
    GO
    ALTER TABLE [dbo].[RiskContext] ADD DEFAULT (getdate()) FOR
[CreatedAt]
    GO
    ALTER TABLE [dbo].[Feature] WITH CHECK ADD FOREIGN
KEY([ID_Task])
    REFERENCES [dbo].[Task] ([ID_Task])
    GO
    ALTER TABLE [dbo].[Feature_Metric] WITH CHECK ADD FOREIGN
KEY([ID_Feature])
    REFERENCES [dbo].[Feature] ([ID_Feature])
    GO
    ALTER TABLE [dbo].[Feature_Metric] WITH CHECK ADD FOREIGN
KEY([ID_Metric])
    REFERENCES [dbo].[Metric] ([ID_Metric])
    GO
    ALTER TABLE [dbo].[Risk] WITH CHECK ADD FOREIGN
KEY([ID_RiskType])
    REFERENCES [dbo].[Risk_Type] ([ID_RiskType])
    GO
    ALTER TABLE [dbo].[Risk] WITH CHECK ADD CONSTRAINT
[FK_SourceRisk_RISK] FOREIGN KEY([ID_SourceRisk])
    REFERENCES [dbo].[SourceRisk] ([ID_SourceRisk])
    GO
    ALTER TABLE [dbo].[Risk] CHECK CONSTRAINT [FK_SourceRisk_RISK]
    GO
    ALTER TABLE [dbo].[Risk_Metric] WITH CHECK ADD FOREIGN
KEY([ID_Metric])
    REFERENCES [dbo].[Metric] ([ID_Metric])
    GO
```

```
    ALTER TABLE [dbo].[Risk_Metric] WITH CHECK ADD FOREIGN
KEY([ID_Risk])
    REFERENCES [dbo].[Risk] ([ID_Risk])
    GO
    ALTER TABLE [dbo].[RiskContext] WITH CHECK ADD FOREIGN
KEY([ID_Feature])
    REFERENCES [dbo].[Feature] ([ID_Feature])
    GO
    ALTER TABLE [dbo].[RiskContext] WITH CHECK ADD FOREIGN
KEY([ID_Project])
    REFERENCES [dbo].[Project] ([ID_Project])
    GO
    ALTER TABLE [dbo].[RiskContext] WITH CHECK ADD FOREIGN
KEY([ID_Risk])
    REFERENCES [dbo].[Risk] ([ID_Risk])
    GO
    ALTER TABLE [dbo].[RiskContext] WITH CHECK ADD FOREIGN
KEY([ID_Sprint])
    REFERENCES [dbo].[Sprint] ([ID_Sprint])
    GO
    ALTER TABLE [dbo].[RiskContext] WITH CHECK ADD FOREIGN
KEY([ID_Task])
    REFERENCES [dbo].[Task] ([ID_Task])
    GO
    ALTER TABLE [dbo].[Sprint] WITH CHECK ADD FOREIGN
KEY([ID_Project])
    REFERENCES [dbo].[Project] ([ID_Project])
    GO
    ALTER TABLE [dbo].[Sprint_Task] WITH CHECK ADD FOREIGN
KEY([ID_Sprint])
    REFERENCES [dbo].[Sprint] ([ID_Sprint])
    GO
    ALTER TABLE [dbo].[Sprint_Task] WITH CHECK ADD FOREIGN
KEY([ID_Task])
    REFERENCES [dbo].[Task] ([ID_Task])
    GO
```

```
ALTER TABLE [dbo].[Task] WITH CHECK ADD FOREIGN KEY([ID_Project])
REFERENCES [dbo].[Project] ([ID_Project])
GO
ALTER TABLE [dbo].[RiskContext] WITH CHECK ADD CONSTRAINT [CHK_RiskContext_OneContextOnly] CHECK (([ContextLevel]='PROJECT' AND [ID_Project] IS NOT NULL AND [ID_Sprint] IS NULL AND [ID_Task] IS NULL AND [ID_Feature] IS NULL OR [ContextLevel]='SPRINT' AND [ID_Project] IS NULL AND [ID_Sprint] IS NOT NULL AND [ID_Task] IS NULL AND [ID_Feature] IS NULL OR [ContextLevel]='TASK' AND [ID_Project] IS NULL AND [ID_Sprint] IS NULL AND [ID_Task] IS NOT NULL AND [ID_Feature] IS NULL OR [ContextLevel]='FEATURE' AND [ID_Project] IS NULL AND [ID_Sprint] IS NULL AND [ID_Task] IS NULL AND [ID_Feature] IS NOT NULL))
GO
ALTER TABLE [dbo].[RiskContext] CHECK CONSTRAINT [CHK_RiskContext_OneContextOnly]
GO
USE [master]
GO
ALTER DATABASE [RISK_MANAGEMENT] SET READ_WRITE
GO
```