ОДЕСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ І. І. МЕЧНИКОВА
(повне найменування закладу вищої освіти)

Факультет математики, фізики та інформаційних технологій
(повне найменування факультету)

Кафедра інформаційних технологій
(повна назва кафедри)

Кваліфікаційна робота

на здобуття ступеня вищої освіти «Магістр»

«Інформаційна система для оптимізації комунікації між стейхолдерами на основі аналізу дерев проєктних рішень за методологією Agile»

(тема кваліфікаційної роботи українською мовою)

«An information system for optimizing communication between stakeholders based on the analysis of project decision trees according to the Agile methodology»

(тема кваліфікаційної роботи англійською мовою)

Виконав: здобувач денної форми навчання
спеціальності 122 Комп'ютерні науки
(код, назва спеціальності)
Освітня програма Комп'ютерні науки
(назва)
ЛАПАЄВ Андрій Валерійович
(прізвище, ім'я, по-батькові здобувача)

Керівник Великодний С. С., д.т.н., доцент _____
(науковий ступінь, вчене звання, прізвище, ініціали) (підпис)

Рецензент Соколов Артем Валерійович, д.т.н., доцент, професор кафедри кібербезпеки НУ "Одеська юридична академія"
(науковий ступінь, вчене звання, прізвище, ініціали)

Рекомендовано до захисту:
Протокол засідання кафедри
Інформаційних технологій
№ ____від _____2025 р.

Завідувачка кафедри

_____ КАЗАКОВА Надія
(підпис) (прізвище,ім'я)

Захищено на засіданні ЕК № _____
протокол № ___від _____2025 р.

Оцінка _____/ _____/ _____
(за національною шкалою/шкалою ECTS/ бали)

Голова ЕК

_____ СОКОЛОВ Артем
(підпис) (прізвище,ім'я)

Одеса 2025

АНОТАЦІЯ

Кваліфікаційна робота присвячена розв’язанню задачі підвищення ефективності планування та управління ризиками в масштабованих Agile-проєктах, що реалізуються за методологією Scaled Agile Framework.

У роботі проаналізовано недоліки наявних методів оцінки трудомісткості (Story Points, Use Case Points) та обґрунтовано необхідність переходу до об’єктивних кількісних метрик. Запропоновано метод оцінки локальної невизначеності програмних артефактів (епіків та фіч) на основі ентропійних дерев рішень, що дозволяє формалізувати структуру технічних альтернатив.

Розроблено математичну модель для аналізу мережевих залежностей між Agile Release Trains з використанням систем лінійних алгебраїчних рівнянь. Це дозволило отримати показник системної невизначеності, який враховує взаємовплив команд у межах програми.

Спроєктовано та реалізовано інформаційну систему у вигляді вебзастосунку на платформі .NET 8 із використанням архітектурного патерну MVC та СКБД PostgreSQL. Система забезпечує автоматизацію побудови дерев рішень, розрахунок ентропії Шеннона та візуалізацію ризиків на аналітичній панелі.

Експериментальна перевірка підтвердила працездатність системи та її ефективність у виявленні прихованих архітектурних ризиків, що дозволяє підвищити точність планування програмних інкрементів.

Ключові слова: SAFe, Agile Release Train, ентропія, дерево рішень, управління ризиками, .NET, MVC.

ABSTRACT

The qualification work is devoted to solving the problem of increasing the efficiency of planning and risk management in scaled Agile projects implemented using the Scaled Agile Framework methodology.

The paper analyzes the shortcomings of existing methods for assessing labor intensity (Story Points, Use Case Points) and justifies the need to switch to objective quantitative metrics. A method for assessing the local uncertainty of software artifacts (epics and features) based on entropy decision trees is proposed, which allows formalizing the structure of technical alternatives.

A mathematical model has been developed for analyzing network dependencies between Agile Release Trains using systems of linear algebraic equations. This allowed obtaining a system uncertainty indicator that takes into account the mutual influence of teams within the program.

An information system has been designed and implemented in the form of a web application on the .NET 8 platform using the MVC architectural pattern and the PostgreSQL database. The system provides automation of decision tree construction, calculation of Shannon entropy and visualization of risks on the analytical panel.

Experimental verification confirmed the system’s operability and its effectiveness in identifying hidden architectural risks, which allows to increase the accuracy of planning software increments.

Keywords: SAFe, Agile Release Train, entropy, decision tree, risk management, .NET, MVC.

ЗМІСТ

ВСТУП

Сучасна індустрія розробки програмного забезпечення характеризується стрімким зростанням масштабу проєктів та переходом великих організацій до використання фреймворків масштабованого Agile, зокрема Scaled Agile Framework. В умовах, коли над продуктом одночасно працюють десятки команд, критичного значення набуває точність планування та оцінки ризиків.

Наявні методи оцінки трудомісткості, такі як Story Points, значною мірою спираються на евристичні судження та експертний досвід. Хоча вони є загальноприйнятим стандартом в індустрії, їх головним недоліком залишається висока суб'єктивність та нездатність кількісно врахувати архітектурні залежності між командами. Помилка в оцінці однієї команди може спричинити каскадний ефект затримок у всій програмі. Тому створення об'єктивних, математично обґрунтованих інструментів для вимірювання невизначеності є актуальним науково-практичним завданням.

Об'єктом дослідження є процеси планування програмних інкрементів та управління ризиками у масштабованих гнучких проєктах, що реалізуються за методологією SAFe.

Предметом дослідження є методи та засоби кількісного оцінювання невизначеності програмних артефактів з використанням ентропійного аналізу та моделювання мережевих залежностей.

Метою роботи є підвищення ефективності управління програмами в середовищі Scaled Agile Framework шляхом розробки інформаційної системи, яка дозволяє об'єктивізувати оцінку ризиків на основі математичного апарату теорії інформації та лінійної алгебри.

Для досягнення поставленої мети розв'язано такі задачі:

а) проведено аналіз принципів Scaled Agile Framework та виявлено недоліки наявних метрик планування;

б) розроблено математичну модель оцінки локальної невизначеності артефактів на основі ентропійних дерев рішень;

в) розроблено метод оцінки системної (мережевої) невизначеності з урахуванням взаємовпливів між Agile Release Trains;

г) спроєктовано архітектуру та схему бази даних інформаційної системи;

д) реалізовано прототип інформаційної системи у вигляді вебзастосунку та експериментально перевірено його ефективність на модельному проєкті.

Методи дослідження. У роботі використано методи системного аналізу (для дослідження предметної області), теорію інформації (для розрахунку ентропії Шеннона), теорію графів та лінійну алгебру (для моделювання та розв'язання систем залежностей), а також об'єктно-орієнтоване проєктування (для розробки ПЗ).

Практичне значення одержаних результатів полягає у створенні працездатного прототипа інформаційної системи, що дозволяє менеджерам (Product Managers, Release Train Engineers) виявляти приховані ризики та оптимізувати плани ще до початку розробки, замінюючи інтуїтивні здогадки точними розрахунками.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Тенденції розвитку індустрії програмного забезпечення характеризуються зростанням масштабу проєктів та підвищенням вимог до адаптивності процесів розробки. Аналіз практичного застосування гнучких методологій свідчить, що, попри їхню ефективність, залишаються невирішеними проблеми суб'єктивності експертних оцінок та втрати інформації при комунікації між стейкхолдерами. Зазначені недоліки набувають критичного значення при використанні масштабованих фреймворків, таких як SAFe [1], де помилки планування призводять до значних економічних втрат. У цьому розділі проаналізовано стан предметної області та обґрунтовано доцільність застосування ентропійного підходу для кількісної оцінки невизначеності проєктних рішень.

Питання оптимізації процесів розробки програмного забезпечення та управління проєктами в умовах невизначеності широко висвітлені у сучасній науковій літературі.

Зокрема, моделі та методи управління життєвим циклом програмного забезпечення детально розглянуто у працях [2, 3]. Питання реінжинірингу та архітектурної оптимізації інформаційних систем досліджено у роботі [4].

Окремої уваги заслуговують дослідження методів математичного моделювання інформаційних процесів, які створюють фундамент для формалізації прийняття рішень [5, 6]. Проте, незважаючи на ґрунтовну теоретичну базу, проблема інтеграції цих математичних моделей у гнучкі методології масштабування (такі як SAFe) для оперативної оцінки ризиків залишається відкритою і потребує розробки нових інструментальних засобів.

1.1 Важливість проєктного менеджменту

Управління IT-проєктами – це процес управління, планування та розробки проєктів у сфері інформаційних технологій. IT-проєкти наявні у різних галузях людської діяльності, наприклад, розробка програмного, апаратного забезпечення, інформаційна безпека, системи, комунікації, мережі, бази даних і мобільні додатки.

Розробники надають продукт або послугу, а проєктні менеджери своєю чергою займаються управлінням IT. Менеджери несуть відповідальність за передачу очікувань і виконання проєктів за графіком та в межах бюджету, щоб забезпечити безперебійну роботу команди [7].

У будь-якому проєкті виділяють основні фази його розвитку [7] та виконання:

– створення, протягом чого визначається потреба в проєкті, створюється проєктна пропозиція та добираються переконливі аргументи щодо доцільності його виконання;

– планування, протягом якого спільними зусиллями та кількісними метриками визначаються розміри проєкту, визначаються ризики та методи їх усунення, обчислюється вартість, створюється дорожня карта;

– виконання, протягом якого команда визначає кінцевий результат проєкту;

– завершення та закриття, коли здобуті знання підсумовуються для подальшої роботи.

Керівник проєктів – це особа, яка знається на спілкування з усіма зацікавленими особами проєкту, відповідає за обговорення роботи та містить низку типових обов’язків [7]:

– забезпечує функціональність продукту через те, що є першою точкою дотику в будь-якій задачі;

– призначає завдання команді;

– відстежує прогрес та продуктивність, часові межі тощо;

– проводить зустрічі між учасниками, у тому числі згідно з принципами гнучких методологій.

Яку б гнучку структуру не обрали б розробники для підтримки програмного забезпечення, їм знадобиться спосіб бачити прогрес своєї команди, щоб вони могли планувати майбутню роботу чи спринт. Гнучка оцінка проєкту допомагає командам Scrum і Kanban зрозуміти їхні можливості. Гнучкі звіти показують прогрес команди з часом [8].

1.2 Необхідність у гнучкому проєктному менеджменті

Виходячи з концепції ощадливого виробництва Toyota 1940-х років, групи розробників програмного забезпечення застосували гнучкі методології для зменшення відходів і підвищення прозорості, одночасно швидко задовольняючи потреби своїх клієнтів, що постійно змінюються. Різка зміна одночасного управління проєктами, яке зосереджується на «великих» запусках, agile допомагає командам програмного забезпечення краще співпрацювати та впроваджувати інновації швидше, ніж будь-коли раніше [8].

Гнучке управління проєктами (Agile) базується на ітеративно-інкрементальній моделі життєвого циклу розробки програмного забезпечення. Ключовими принципами цього підходу є регулярне постачання робочих версій продукту та адаптація до змінних вимог замовника через механізми зворотного зв'язку.

Серед методів реалізації Agile найбільш поширеними є фреймворки Scrum та Kanban. Якщо Scrum структурує процес за допомогою часових інтервалів фіксованої довжини (спринтів), то Kanban фокусується на безперервному потоці виконання завдань (Flow-based), де нові задачі беруться в роботу безпосередньо після завершення попередніх, що дозволяє оптимізувати пропускну здатність команди.

Практичне застосування гнучких методологій дозволяє скоротити час виведення продукту на ринок (Time-to-Market), підвищити ефективність внутрішньокомандної взаємодії та забезпечити стійкість проєкту до змін ринкової кон'юнктури [8].

1.3 Взаємодія учасників

Ефективність реалізації гнучких проєктів безпосередньо корелює з якістю управління зацікавленими сторонами (стейкхолдерами). На відміну від традиційних каскадних моделей, Agile-підхід вимагає безперервної взаємодії та адаптації комунікаційних стратегій, що створює специфічні управлінські виклики.

Фундаментом побудови комунікаційної матриці проєкту є класифікація стейкхолдерів за ступенем впливу та походженням. За рівнем залученості виділяють:

– первинних стейкхолдерів (Product Owner, команда розробників, кінцеві користувачі), які безпосередньо формують вимоги до функціональності продукту;

– вторинних стейкхолдерів (регуляторні органи, підрозділи підтримки, топ-менеджмент), які визначають обмеження, стандарти відповідності (compliance) та інтеграційні вимоги.

Окрім цього, важливим є розмежування на внутрішніх (співробітники організації) та зовнішніх (клієнти, партнери, постачальники) учасників. Ця диференціація визначає вибір каналів зворотного зв'язку: для внутрішньої комунікації ефективними є міжфункціональні воркшопи, тоді як робота із зовнішніми контрагентами вимагає формалізованих сесій огляду продукту (Review) та узгодження очікувань.

Невідповідність комунікаційних стратегій динаміці проєктного середовища може призвести до негативних наслідків, зокрема [9]:

– розсинхронізація очікувань між первинними та вторинними групами загрожує затримками у постачанні інкрементів;

– недостатнє залучення зовнішніх стейкхолдерів підвищує ризик створення продукту, що не відповідає потребам ринку;

– ігнорування вторинних внутрішніх команд на ранніх етапах ускладнює процеси впровадження та підтримки.

В умовах Agile стейкхолдери виступають не пасивними спостерігачами, а активними учасниками формування змісту (Scope) та пріоритетів проєкту. Регулярні ітеративні огляди (Sprint Reviews) дозволяють мінімізувати ентропію вимог шляхом раннього виявлення відхилень від бізнес-цілей. Ключовим завданням менеджера стає не лише ідентифікація всіх учасників процесу, але й забезпечення об'єктивності та прозорості інформаційних потоків для усунення комунікаційних розривів.

1.4 Методи гнучкого управління проєктами

Scrum – це структура для гнучкого управління проєктами, яка використовує ітерації роботи фіксованої довжини, які називаються спринтами. Усе починається з беклогу [8], яких існує два різновиди:

– беклог продукту, який належить власнику продукту та є пріоритетним списком функцій;

– беклог спринту, який заповнюється шляхом розгляду проблем із верхньої частини беклогу продукту до потенціалу для наступного спринту досягнуто.

Типові ітерації Scrum [8] зазвичай містять у собі елементи, які показані у табл. 1.1.

Таблиця 1.1 – Чотири ітерації Scrum

Планування	Демонстрація	Стендап	Ретроспектива
Нарада з планування команди, яка визначає, що треба завершити в майбутньому спринті	Зустріч для спільного використання, на якій команда показує, що вони відправили під час спринту	15-хвилинна мінізустріч для команди програмного забезпечення з метою синхронізації	Огляд того, що вдалось, що – ні, щоб зробити наступний спринт кращим

Дошка scrum використовується для візуалізації всієї роботи в даному спринті. Під час зустрічі з планування спринту команда переміщує елементи з резерву продукту в резерв спринту. Дошки Scrum можуть мати кілька видимих кроків у робочому процесі, як-от зробити, у процесі, зроблено. Дошки Scrum є ключовим компонентом для підвищення прозорості гнучкого управління проєктами [8].

Kanban – це структура для гнучкого управління проєктами, яка узгоджує роботу з можливостями команди. Він зосереджений на тому, щоб виконувати завдання якомога швидше, надаючи командам можливість реагувати на зміни навіть швидше, ніж сутички [8].

На відміну від Scrum, який структурує роботу за допомогою фіксованих ітерацій (спринтів), Kanban не використовує резерви завдань, прив’язані до часового інтервалу. Натомість, робочий процес організований за принципом безперервного потоку (Flow), де всі завдання візуально представлені на дошці, починаючи зі стовпця Backlog. Це дає змогу команді зосередитися на безперервному постачанні продукту (Continuous Delivery). Головна особливість методології полягає в тому, що команда переходить до наступного завдання одразу після завершення поточного. Обсяг роботи, яку одночасно виконує команда, узгоджується з її можливостями

через WIP-ліміти (Work In Progress limits) – попередньо визначені обмеження, що встановлюють максимальну кількість завдань, які можуть перебувати в одному стовпці дошки (за винятком загального Backlog).

Структура канбану [8] містить компоненти, які показані у табл. 1.2.

Таблиця 1.2 – Компоненти Kanban

Перелік робіт	Відповідні колонки дошки	Ліміти	Випуски
Список робіт, або історії, визначаються як питання або завдання, які потрібно виконати	Існує канбан дошка, щоб розрізняти завдання від різних робочих потоків, користувачів, проєктів тощо	Правило обмеження обсягу роботи, яку необхідно виконати, залежно від можливостей команди	Команда працює над історіями у межах ліміту і може опублікувати будь-коли

Постійне зростання масштабу проєктів та збільшення кількості залучених команд зумовлюють необхідність адаптації гнучких методологій. В таких умовах традиційні підходи Agile, орієнтовані на невеликі, автономні команди, втрачають ефективність, оскільки не можуть гарантувати необхідної узгодженості та синхронізації. Одним із ключових інструментів, розроблених для нівелювання цієї проблеми, є Scaled Agile Framework (SAFe). SAFe надає структурований та комплексний підхід до реалізації Agile-практик, забезпечуючи ефективну взаємодію на рівнях програми та портфеля великих організацій [1].

SAFe об’єднує принципи Lean, Agile та DevOps, організовуючи роботу на кількох рівнях для забезпечення синхронізації та узгодженості. На рівні команди (Team Level) функціонують окремі Agile-команди, що використовують Scrum, Kanban або гібридні підходи для виконання користувацьких історій. На програмному рівні (Program Level) робота масштабується через Agile Release Trains (ARTs) – віртуальні організації «команд команд», які спільно постачають цінність у рамках програмних інкрементів (PIs). Програмний інкремент – це фіксований проміжок часу (зазвичай 8–12 тижнів), протягом якого ART створює інкрементальний, протестований приріст рішення. Центральною подією кожного PI є PI Planning (планування програмного інкременту), дводенна зустріч, де всі команди ART, зацікавлені сторони

та керівництво збираються для спільного планування, ідентифікації залежностей, обговорення ризиків та формування зобов'язань на наступний PI.

Вище програмного рівня може існувати рівень великих рішень (Large Solution Level) для координації роботи кількох ARTs над дуже великими та складними системами, а на рівні портфеля (Portfolio Level) відбувається стратегічне фінансування, визначення цінності та узгодження великих ініціатив (епіків) з бізнес-цілями компанії. Такий багаторівневий підхід дозволяє SAFe синхронізувати діяльність багатьох команд та спрямовувати її на досягнення стратегічних цілей бізнесу, забезпечуючи при цьому прозорість та гнучкість на кожному етапі розробки.

Незважаючи на регламентованість процесів координації у SAFe, масштабування Agile-практик виявляє обмеження існуючих методів оцінювання. Ключовою проблемою залишається висока залежність від евристичних підходів та експертних суджень при визначенні складності артефактів (епіків та фіч). Суб'єктивність локальних оцінок на рівні команд призводить до кумулятивної похибки на рівні програми, що спричиняє розсинхронізацію планів між взаємозалежними Agile Release Trains.

Комунікаційна складність у розподілених середовищах створює бар'єри для об'єктивного сприйняття ризиків та архітектурних залежностей. Відсутність формалізованих кількісних метрик невизначеності ускладнює прийняття рішень на рівні управління портфелем та знижує точність прогнозування термінів реалізації. Це зумовлює необхідність впровадження інструментарію, що забезпечує перехід від інтуїтивних оцінок до математично обґрунтованих показників для верифікації проєктних рішень.

1.5 Технічні рішення для взаємодії учасників проєкту

У сфері гнучкого управління проєктами ключем до успіху є адаптивність, співпраця та ітераційний прогрес. З огляду на ці цілі, групи розробників програмного забезпечення створили філософію та структуру Scrum. Scrum базується на ідеї спільної роботи команд для досягнення спільної мети. З моменту свого створен-

ня scrum став широко використовуваним фреймворком для вирішення складних завдань, заохочення інновацій та надання цінності.

Дошка Scrum – це гнучкий інструмент управління проєктами, який дозволяє командам, які прагнуть працювати поступово, візуалізувати, відстежувати та керувати роботою протягом спринту або фіксованого періоду часу. Його структура заохочує чітке спілкування, прозорі робочі процеси та оптимізоване керування завданнями.

Дошка Scrum візуально представляє принципи гнучкості. Кожна картка на дошці – це робочий елемент, який команда визначила пріоритетом для цього спринту. Разом колонки на дошці складають робочий процес команди. Коли командна робота просувається вперед, картки на дошці також рухаються. Усі інші робочі елементи, які не були пріоритетними для цього спринту, знаходяться в резерві scrum [10]. На рис. 1.1 показано приклад такої Scrum [10] дошки.

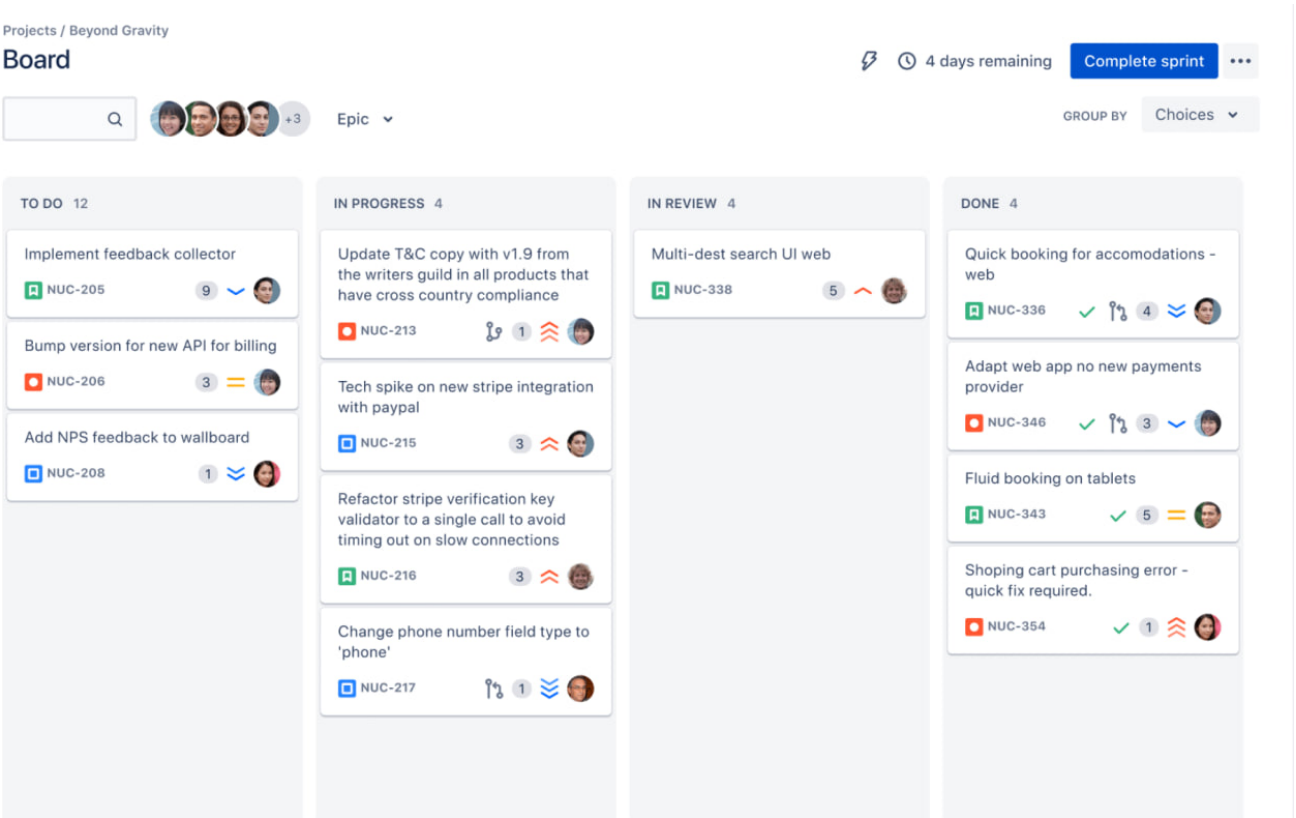

Рисунок 1.1 – Приклад Scrum дошки

Інший відомий ресурс Miro [11] також має реалізацію Scrum дошки з наступними можливостями:

– компоновані робочі процеси: дозволяє легко створювати автоматизовані робочі процеси Scrum, комбінуючи інтегровані віджети, такі як додаток Dependencies and Estimations, щоб візуалізувати залежності та оцінки зусиль безпосередньо на дошці Scrum, без необхідності перемикатися між інструментами;

– повна інтеграція: Miro інтегрується з більш ніж 150 програмами, включаючи двосторонню синхронізацію з Jira та Azure DevOps, що дозволяє легко імпортувати історії користувачів, епіки та завдання з інших інструментів і підтримувати єдиний огляд усіх робочих процесів спринту;

– використання інтелектуальних шаблонів: платформа надає набір попередньо конфігурованих макетів для швидкого розгортання робочого простору Scrum; застосування спеціалізованих шаблонів планування спринту дозволяє автоматизувати процес налаштування середовища, забезпечуючи концентрацію зусиль команди на структуруванні беклогу продукту та визначенні цілей ітерації [11].

Вигляд Scrum дошки у реалізації Miro [11] показано на рис. 1.2.

Ресурс Miro пропонує реалізацію канбан дошки з наступними можливостями:

– швидка візуалізація процесів: наявні потужні картки Miro, щоб додавати завдання лише кількома клацаннями та витрачати свій час на складні робочі процеси, використовуючи налаштування за замовчуванням або налаштувавши дошку Kanban відповідно до власних потреб;

– гнучке створення: дошка Miro має гнучку структуру, у якій можна перетягувати завдання, і дошка автоматично переформатує картки, або налаштувати свій інструмент Kanban, позначивши завдання кольором, додавши більше стовпців і доріжок, усе відповідно до потреб робочого процесу;

– оптимізація взаємодії зі стейкхолдерами: забезпечення спільного доступу до візуального простору дозволяє консолідувати зауваження всіх учасників проєкту; механізми миттєвого зворотного зв’язку через систему коментарів та графічних анотацій безпосередньо на Kanban-дошці сприяють скороченню циклів ітеративного вдосконалення продукту.

Вигляд реалізації канбан дошки у Miro показано на рис. 1.3.

Gitlub Issue Board [12] – це інструмент керування програмним проєктом, який використовується для планування, організації та візуалізації робочого процесу для

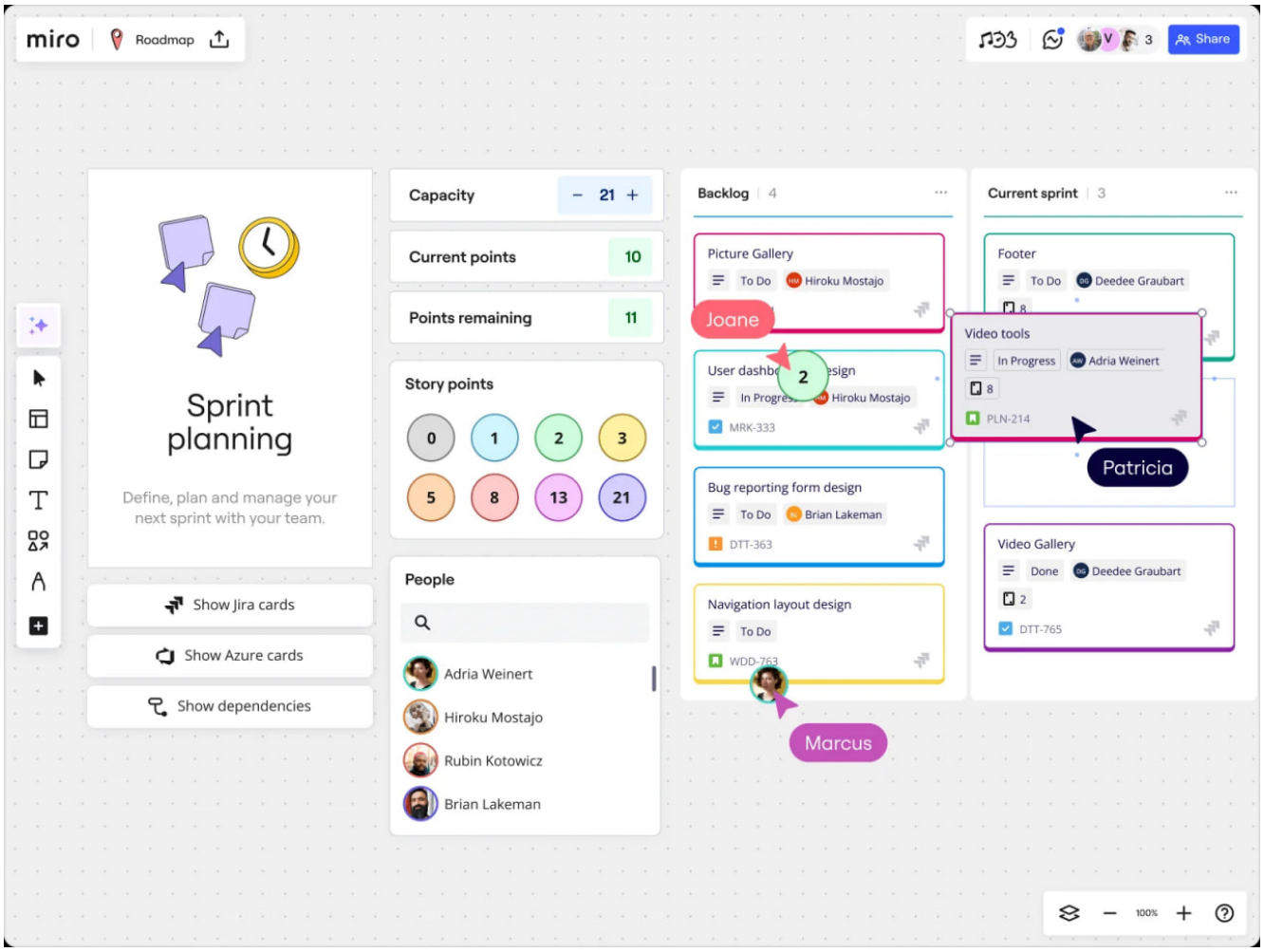

Рисунок 1.2 – Scrum дошка з Miro

випуску функції чи продукту. Його можна використовувати його як дошку Kanban або Scrum.

Інструмент дошок завдань (Issue Boards) забезпечує інтеграцію механізмів відстеження проблем та управління проєктами, що дозволяє централізувати робочий процес та організувати його в межах єдиної платформи.

Дошки випусків використовують випуски та етикетки. Проблеми показані у вигляді карток у вертикальних списках, упорядкованих за призначеними мітками, віхами або особами, які їм призначені.

Дошки проблем допомагають візуалізувати та керувати всім процесом у GitLab. Користувачі додають свої мітки, а потім створюють відповідний список для наявних проблем.

Дошка питань може показати проблеми, над якими працює команда, хто призначений для кожної з них і де проблеми знаходяться в робочому процесі. Щоб

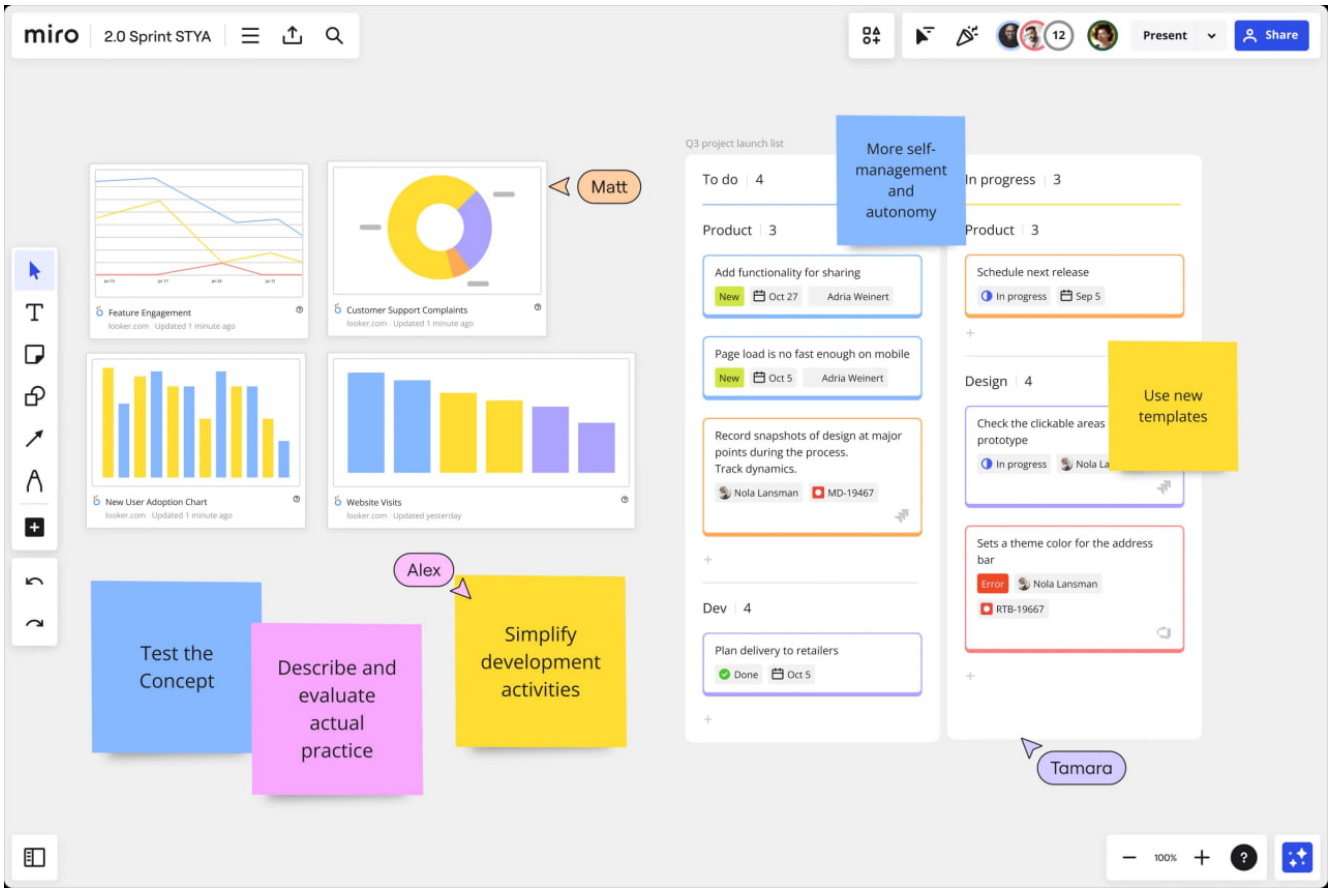

Рисунок 1.3 – Можливості Miro Kanban дошки

дозволити членам команди організовувати власні робочі процеси, можна використати кілька дощок питань. Це дозволяє створювати кілька дощок проблем в одному проєкті.

Зовнішній вигляд такої [12] дошки показано на рис. 1.4.

Зі зростанням масштабів проєктів та необхідністю застосування фреймворків, таких як Scaled Agile Framework [1], виникає потреба в адаптації цих інструментів для підтримки команд на рівнях програми та портфеля. Платформи, подібні до Jira (яку підтримує Miro [11, 13]) та GitLab [12], активно використовуються для реалізації практик SAFe, дозволяючи налаштовувати дошки для управління епіками, можливостями та залежностями між Agile Release Trains (ARTs). Вони надають функціонал для візуалізації потоку робіт через різні рівні, підтримки PI Planning та відстеження прогресу на рівні програми.

Незважаючи на широкі можливості адаптації, наявні програмні рішення мають суттєві обмеження в частині об’єктивізації оцінки проєктних ризиків. Хоча сучасні інструменти ефективно забезпечують візуалізацію робочих процесів та базову

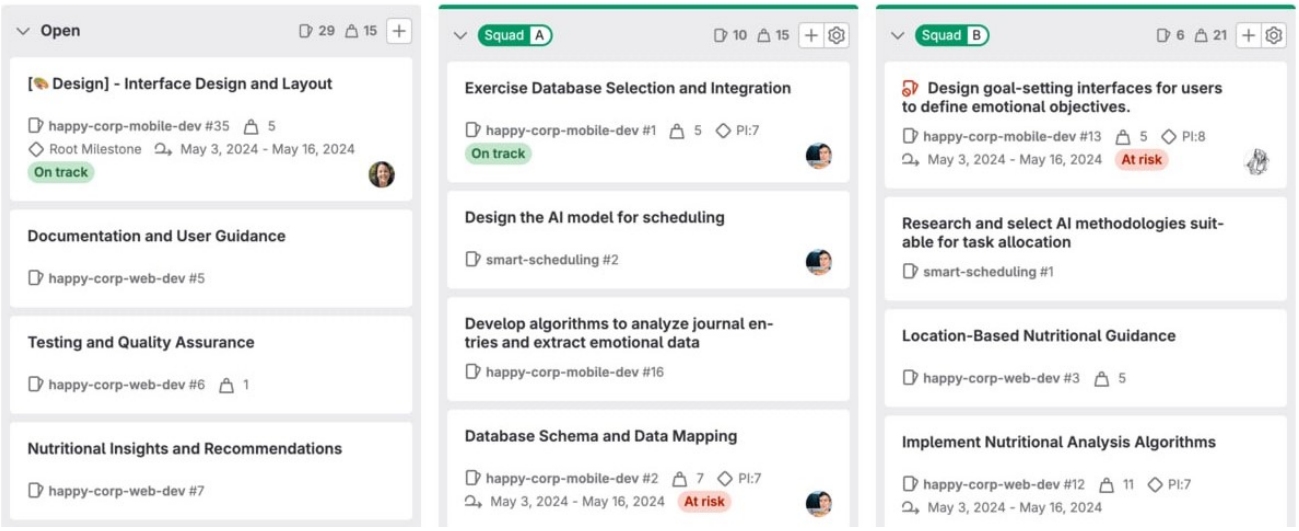

Рисунок 1.4 – Можливості Gitlub Issue дошки

координацію команд, вони часто позбавлені вбудованих механізмів для кількісного вимірювання невизначеності, зумовленої масштабуванням системи.

Відсутність формалізованих метрик ускладнює прозору комунікацію між стейкхолдерами на рівнях програми та портфеля, зокрема при узгодженні неочевидних залежностей. Акцент наявних засобів на візуалізації поточного стану (status reporting) залишає поза увагою аналіз причинно-наслідкових зв’язків складності, що є критичним для проактивного управління ризиками. Це обґрунтовує необхідність розширення функціональності інструментарію аналітичними модулями, здатними підвищити точність та об’єктивність інформаційного забезпечення учасників проєкту.

2 МЕТОДИ ТА МОДЕЛІ КІЛЬКІСНОГО ОЦІНЮВАННЯ НЕВИЗНАЧЕНОСТІ У МАСШТАБОВАНИХ AGILE-СИСТЕМАХ

Висновки попереднього розділу вказують на критичну потребу в удосконаленні процесів комунікації та оцінювання в межах масштабованих фреймворків, таких як SAFe. Наявний інструментарій не забезпечує належного рівня об’єктивності при роботі з невизначеністю, що вимагає залучення нових підходів. У цьому розділі здійснено аналіз математичних методів кількісної оцінки, а також розглянуто технологічні та архітектурні рішення, що формують теоретичне підґрунтя для розробки запропонованої інформаційної системи.

2.1 Кількісні показники в Agile

Принципи Agile [8] записані у документі, який називається маніфестом. Хоча ці принципи містять лише висловлення, побажання та якісні показники, на практиці команди використовують й кількісні, для точної оцінки та виміру.

2.1.1 Бальна оцінка варіантів використання

При плануванні виконання проєкту потрібно обчислити його складність, вартість тощо. Одним із відомих варіантів обчислення складності є метод бальної оцінки варіантів використання.

Якщо команда працювала з варіантами використання, вона, напевно, відчувала, що має бути простий спосіб оцінити загальний розмір проєкту на основі всієї роботи, яка була проведена для написання варіантів використання. Існує чіткий зв’язок між варіантами використання та кодом, оскільки кодування складних варіантів використання зазвичай займає більше часу, ніж простих. Наразі існує підхід для оцінки та планування з використанням варіантів використання. Подібні за концепцією до функціональних точок, точки використання вимірюють розмір програми. Коли ми знаємо приблизний розмір програми, ми можемо визначити очі-

кувану тривалість проєкту, якщо ми також знаємо (або можемо оцінити) швидкість прогресу команди.

Кількість точок варіантів використання в проєкті є функцією наступного:

– кількість і складність варіантів використання в системі;

– кількість і складність акторів у системі;

– різні нефункціональні вимоги, які не написані як варіанти використання;

– середовище, в якому буде розвиватися проєкт.

Методика розрахунку підсумкового показника точок варіантів використання (Use Case Points) базується на агрегації зважених оцінок складності прецедентів та акторів. Отримане значення нескоригованого розміру системи надалі підлягає калібруванню за допомогою коефіцієнтів, що відображають вплив нефункціональних вимог та факторів середовища розробки.

Фундаментальним для використання пунктів варіантів використання є необхідність, щоб усі варіанти використання були написані приблизно на одному рівні. Визначається п’ять рівнів для варіантів використання: дуже високий підсумок, підсумок, мета користувача, підфункція та занадто низький. Дуже високий підсумок і зведені варіанти використання корисні для встановлення контексту, у якому працюють варіанти використання нижчого рівня. Однак вони написані на надто високому рівні, щоб бути корисними для оцінки. Рекомендується, щоб випадки використання на рівні цілей користувача становили основу добре продуманої колекції варіантів використання. На нижчому рівні сценарії використання підфункції написані для надання деталей за потреби.

Якщо команда проєкту бажає зробити оцінку за допомогою точок варіантів використання, вони повинні написати свої варіанти використання на рівні цілей. Кожен варіант використання має мету. Мета сценарію використання на рівні цілі користувача є фундаментальною одиницею бізнес-цінності [14].

Метод визначення оцінки розміру для розробки системи базується на розрахунку [14] з такими елементами:

– нескоригована вага варіантів використання (UUCW) – розмір точки програмного забезпечення, що враховує кількість і складність варіантів використання;

– нескоригована вага актора (UAW) – розмір точки програмного забезпечення, що враховує кількість і складність акторів;

– коефіцієнт технічної складності (TCF) – коефіцієнт, який використовується для коригування розміру на основі технічних міркувань;

– фактор екологічної складності (ECF) – коефіцієнт, який використовується для коригування розміру на основі екологічних міркувань.

Після обчислення попередніх чотирьох елементів можна розрахувати остаточну оцінку розміру. Це остаточне число відоме як точки використання або UCP для проєкту розробки програмного забезпечення.

Нескоригована вага варіанту використання $UUCW$ [14] є одним із факторів, які впливають на розмір програмного забезпечення, що розробляється. Він розраховується на основі кількості та складності варіантів використання системи. Щоб знайти $UUCW$ для системи, кожен з варіантів використання має бути ідентифікований і класифікований як простий, середній або складний на основі кількості транзакцій, які містить варіант використання. Кожна класифікація має попередньо визначену вагу. Коли всі варіанти використання класифікуються як прості, середні або складні, загальна вага ($UUCW$) визначається шляхом підсумовування відповідних ваг для кожного варіанту використання. Класифікація складності проєкту залежно від кількості транзакцій та відповідна вага $UUCW$ наведено у табл. 2.1.

Таблиця 2.1 – Класифікація складності залежно від кількості транзакцій

Класифікація варіантів використання	Кількість транзакцій	Вага
Простий	1–3	5
Середній	4–7	10
Складний	> 8	15

Числове значення $UUCW$ обчислюється як сума добутків складників на відповідну їм вагу.

UAW [14] є ще одним фактором, який впливає на розмір програмного забезпечення, що розробляється. Він розраховується на основі кількості та складності акторів для системи. Подібно до розрахунку $UUCW$, кожен з учасників має бути

ідентифікований і класифікований як простий, середній або складний залежно від його типу. Кожна класифікація також має попередньо визначену вагу. UAW – це загальна сума ваг для кожного з учасників. У табл. 2.2 показано різні класифікації акторів і присвоєне значення ваги.

Таблиця 2.2 – Класифікація складності залежно від типу

Класифікація	Тип	Вага
Простий	Зовнішня система, яка повинна взаємодіяти з системою за допомогою чітко визначеного API	1
Середній	Зовнішня система, яка повинна взаємодіяти з системою за допомогою стандартних протоколів зв'язку	2
Складний	Користувач використовує графічний інтерфейс програми	3

Числове значення показника UAW обчислюється аналогічно $UUCW$.

TCF [14] є одним із факторів, що застосовуються до розрахункового розміру програмного забезпечення для врахування технічних міркувань системи. Він визначається шляхом присвоєння оцінки від 0 (фактор не має значення) до 5 (фактор важливий) кожному з 13 технічних факторів, перелічених у табл. 2.3. Потім ця оцінка множиться на визначене зважене значення для кожного фактора. Сума всіх розрахованих значень є технічним фактором (TF). Потім TF використовується для обчислення TCF за формулою (2.1).

$$TCF = 0{,}6 + \frac{TF}{100} \tag{2.1}$$

ECF [14] є ще одним фактором, який застосовується до розрахункового розміру програмного забезпечення, щоб врахувати екологічні міркування системи. Він визначається шляхом присвоєння оцінки від 0 (немає досвіду) до 5 (експерт) кожному з 8 факторів навколишнього середовища, перелічених у табл. 2.4. Потім ця оцінка множиться на визначене зважене значення для кожного фактора. Сума всіх обчислених значень є фактором середовища (EF). Потім EF використовується

Таблиця 2.3 – Класифікація складності залежно від фактора

Фактор	Пояснення	Вага
T1	Система розподілена	2
T2	Час відгуку	1
T3	Ефективність користувача	1
T4	Складність обробки	1
T5	Повторне використання коду	1
T6	Легкість встановлення	0,5
T7	Легкість використання	0,5
T8	Портативність	2
T9	Обслуговування	1
T10	Паралельна обробка	1
T11	Безпека	1
T12	Доступ третіх осіб	1
T13	Навчання кінцевого користувача	1

для обчислення ECF за формулою (2.2).

$$ECF = 1{,}4 + (-0{,}03 \times EF) \tag{2.2}$$

Зрештою, UCP [14] можна розрахувати після того, як визначено нескоригований розмір проєкту ($UUCW$ та UAW), технічний фактор (TCF) і фактор середовища (ECF). UCP розраховується за формулою (2.3).

$$UCP = (UUCW + UAW) \times TCF \times ECF \tag{2.3}$$

Це був один із наявних та відомих методів обчислення складності проєкту – бальною оцінкою варіантів використання.

2.1.2 Story points

У гнучкій розробці власнику продукту доручається визначити пріоритетність резерву – упорядкованого списку робіт, який містить короткий опис усіх бажаних

Таблиця 2.4 – Класифікація складності залежно від середовища

Фактор	Опис	Вага
E1	Знайомство з використаним процесом розробки	1,5
E2	Досвід застосування	0,5
E3	Об’єктноорієнтований досвід команди	1
E4	Можливості провідного аналітика	0,5
E5	Мотивація команди	1
E6	Стабільність вимог	2
E7	Позаштатний персонал	-1
E8	Складна мова програмування	-1

функцій і виправлень для продукту. Власники продукту вловлюють вимоги бізнесу, але не завжди розуміють деталі впровадження. Таким чином, хороша оцінка може дати власнику продукту нове уявлення про рівень зусиль для кожного робочого елемента, що потім повертається до його оцінки відносного пріоритету кожного елемента.

Коли команда інженерів починає процес оцінювання, зазвичай виникають запитання щодо вимог та історій користувачів. І це добре: ці запитання допомагають усій команді краще зрозуміти роботу. Спеціально для власників продуктів розбиття робочих елементів на деталізовані частини та оцінки за допомогою сюжетних точок допомагає їм визначити пріоритетність усіх (і потенційно прихованих) областей роботи. І як тільки вони отримають оцінки від команди розробників, власник продукту нерідко змінює порядок елементів у беклозі.

Ключовим є залучення всіх працівників до команди. Кожен член команди по-різному бачить продукт і роботу, необхідну для створення історії користувача. Наприклад, якщо керівництво продукту хоче зробити щось, що здається простим, наприклад, підтримка нового веббраузера, розробка та контроль якості мають зважити, оскільки їхній досвід навчив їх, які дракони можуть ховатися під поверхнею.

Подібним чином зміни дизайну вимагають не лише внеску команди дизайнерів, але й розробки та контролю якості. Залишення частини ширшої команди продукту поза процесом оцінювання створює нижчі оцінки якості, знижує моральний дух,

оскільки ключові учасники не відчувають себе включеними, і погіршує якість програмного забезпечення.

Традиційні підходи до розробки програмного забезпечення зазвичай оперують абсолютними часовими оцінками (людино-години, дні). Натомість у гнучких методологіях стандартом стала оцінка в умовних відносних одиницях – Story Points. Story Points визначаються як міра сукупних зусиль, необхідних для повної реалізації елемента беклогу продукту.

Присвоєння оцінки базується на комплексному аналізі трьох факторів: технічної складності задачі, обсягу роботи та рівня невизначеності (ризиків). Використання абстрактних величин дозволяє команді уникнути хибної точності, притаманної часовим прогнозам, та абстрагуватися від навичок окремих виконавців. Такий підхід сприяє досягненню консенсусу всередині команди та дозволяє більш точно прогнозувати швидкість роботи (Velocity) на основі емпіричних даних попередніх ітерацій [15]. Ось кілька причин використовувати історійні точки:

– дати не враховують роботу, не пов’язану з проєктом, яка неминуче вкрадається в наші дні: електронні листи, зустрічі та співбесіди, до яких може бути залучений член команди;

– кожна команда оцінюватиме роботу за дещо іншою шкалою, що означає, що їхня швидкість (вимірюється в балах), природно, буде різною, що унеможливлює політику, використовуючи швидкість;

– оптимізація процесу оцінювання: після калібрування шкали відносних оцінок та досягнення консенсусу щодо базових значень, команда отримує можливість оперативно визначати трудомісткість нових завдань, мінімізуючи витрати часу на додаткові узгоджувальні процедури.

Команди, починаючи з сюжетних моментів, використовують вправу під назвою планування покеру. Команда візьме пункт із беклогу, коротко обговорить його, і кожен член подумки сформулює оцінку. Потім кожен обирає картку з числом, яке відображає його оцінку. Якщо всі згодні, чудово! Якщо ні, треба виділити деякий час (але не надто багато – лише пару хвилин), щоб зрозуміти обґрунтування різних оцінок. Однак варто пам’ятати, що оцінювання має бути діяльністю високого рівня.

Якщо команда зайшла занадто далеко, треба перевести подих і перевести дискусію на вищий рівень [15].

Уже відомий ресурс Miro [16] дозволяє автоматизувати викладання карток з наступними можливостями:

– дозволяє легке оцінювання: воно відбувається декількома клацаннями мишкою;

– консолідація інформаційного простору: платформа забезпечує створення єдиного візуального середовища, що дозволяє об’єднати інструменти фасилітації (віртуальні стікери, картки), інтегровані завдання із зовнішніх трекінгових систем (Jira) та супровідну проєктну документацію (графічні схеми, макети дизайну);

– надає змістовну співпрацю: процес проведення стає інклюзивнішим, дає кожному можливість проголосувати;

– можливість повідомити, коли учасники готові;

– можливість переглянути хто з учасників не згоден;

– синхронізація з іншими програмними засобами Miro.

Зовнішній вигляд такого [16] застосунку показано на рис. 2.1 , 2.2.

2.1.3 Порівняльний аналіз та виявлені недоліки наявних методів оцінки

Незважаючи на те, що методи UCP та Story Points сприяли формалізації процесів оцінювання трудомісткості, їх практичне застосування виявляє низку системних обмежень. Зазначені недоліки суттєво знижують достовірність прогнозів та ефективність планування, особливо в умовах високої невизначеності та динамічного масштабування проєктів.

Оцінки в обох методах ґрунтуються на експертних судженнях. UCP формалізує процес, але вихідні дані (класифікація акторів, оцінки факторів TCF/ECF) все одно залежать від людського чинника. Story Points відкрито покладаються на консенсус команди, що робить їх вразливими до групових упереджень.

Жоден із розглянутих підходів не забезпечує інструментарію для кількісного вимірювання ступеня невизначеності, зумовленої варіативністю технічних рішень, що імпліцитно містяться у вимогах. Традиційні методики фокусуються на оцінці

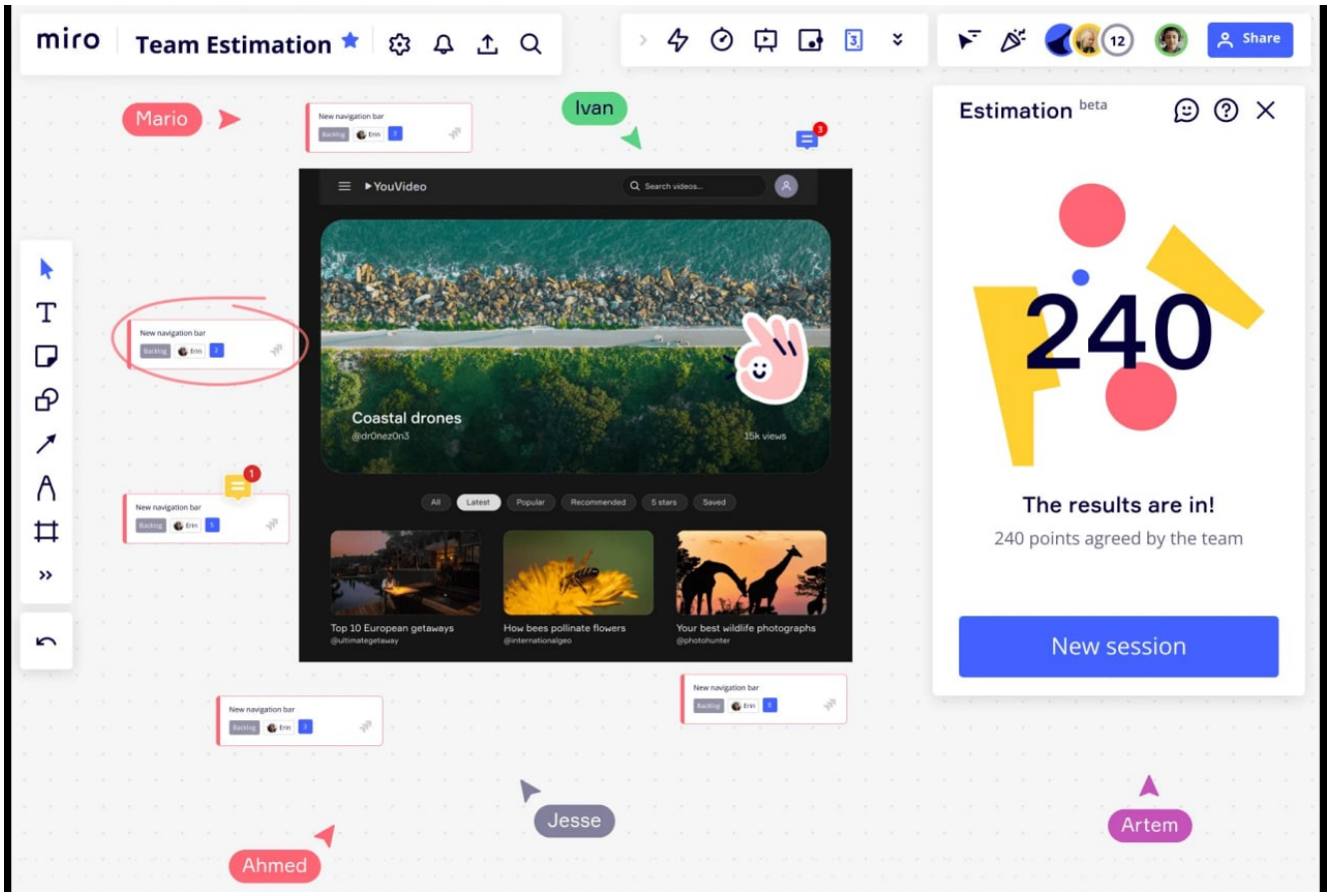

Рисунок 2.1 – Можливості Miro Planning Pocker

трудомісткості або розміру функціоналу, залишаючи поза увагою інформаційну ентропію – об’єктивну характеристику неоднозначності вибору шляхів реалізації проєкту.

Story Points унікальні для кожної команди, що унеможливлює об’єктивне порівняння епіків між різними командами в межах ART у SAFe. UCP, хоча й більш універсальний, важко адаптується до нефункціональних вимог та швидкої зміни умов, характерних для Agile.

Щобільше, ці методи фіксують результат оцінки, але не фіксують та не структурують логіку та альтернативи проєктних рішень, що призвели до такої оцінки. Це ускладнює відстеження змін та аналіз впливу.

Таким чином, недоліками наявних рішень є:

– суб’єктивність та залежність від досвіду;
– відсутність кількісної міри невизначеності;
– складна масштабованість та порівнянність;
– відрив від причинно-наслідкових зв’язків.

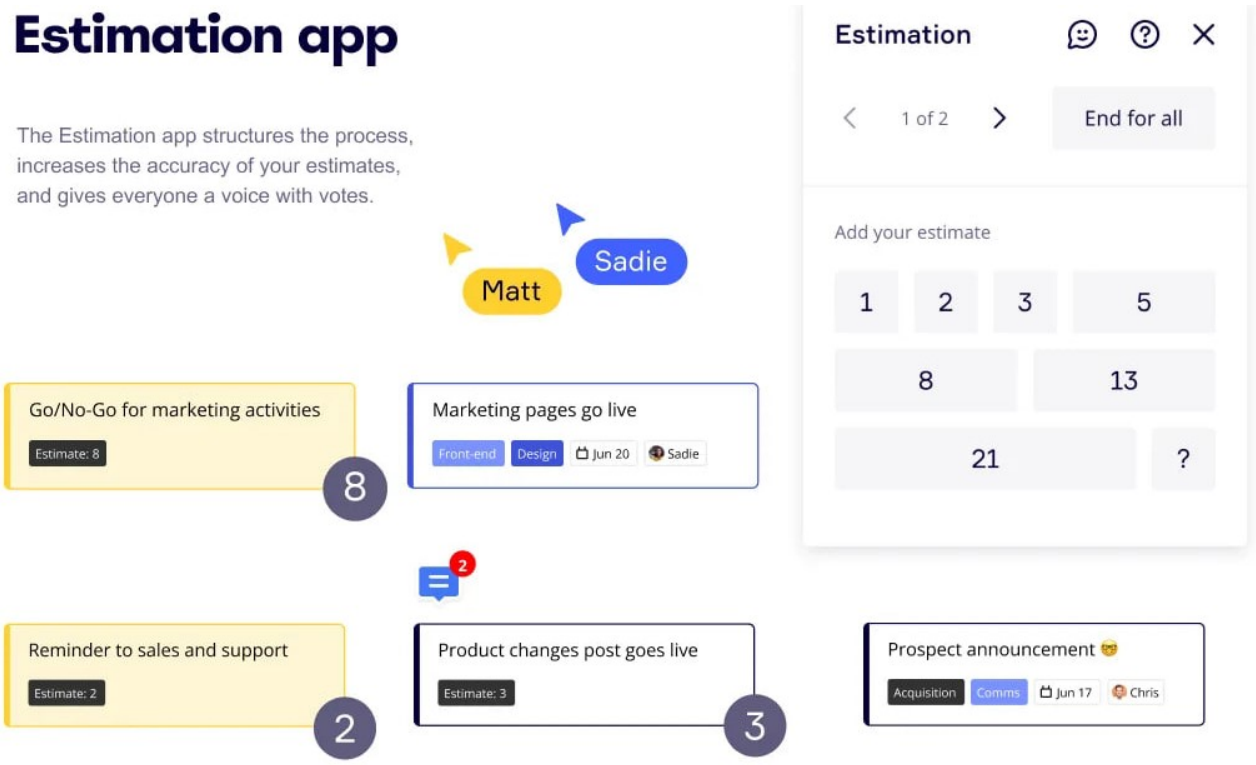

Рисунок 2.2 – Можливості Miro Story Points

Ці недоліки істотно обмежують здатність до ефективного планування в умовах високої невизначеності та масштабованої розробки. Виникла потреба в методі, який би:

– надавав кількісну, об’єктивну міру невизначеності вимоги;

– враховував би структуру проєктних рішень та їх залежності;

– був би масштабованим та корисним для комунікації між різними рівнями управління (команда, програма, портфель).

Саме ці потреби обґрунтовують розробку інформаційної системи на основі аналізу ентропійних дерев рішень, яка запропонована в даній роботі.

2.2 Ентропійна модель невизначеності

Ентропія – провідне поняття у теорії інформації, яке вперше представлене Клодом Шенноном у виданні A Mathematical Theory of Communication 1948 року [17].

Ентропія є мірою невизначеності та непередбачуваності випадкової величини, або мірою середньої кількості інформації, що міститься у повідомленні. На тлі

управління проєктами, ентропія дозволяє отримати кількісну оцінку ризику або невизначеності, пов'язаної з настанням певних подій. За означенням, чим вище значення ентропії, тим більша невизначеність щодо перебігу подій, і, відповідно, вищий імовірний ризик для успішності проєкту.

Для дискретної випадкової величини X з множиною можливих результатів $x_1, x_2, \dots, x_n$ та відповідними ймовірностями $p(x_1), p(x_2), \dots, p(x_n)$, ентропія Шеннона $H(X)$ визначається за формулою (2.4) [18].

$$H(X) = -\sum_{i=1}^{n} p(x_i) \log_b p(x_i), \tag{2.4}$$

де $p(x_i)$ — ймовірність настання i-го результату, сума яких $\sum_{i=1}^{n} p(x_i) = 1$;
$\log_b p(x_i)$ — логарифм ймовірності, основа якого вибирається залежно від поставленої задачі.

Якщо для $\forall i$ ймовірність $p(x_i) = 0$, то відповідний доданок $p(x_i) \log_b p(x_i)$ за домовленістю [18] вважається рівним нулю, оскільки $\lim_{p \to 0^+} p \log p = 0$.

Очевидно, що аналіз проєктних рішень передбачає присутність виключно дискретних випадкових величин (імовірних причинно-наслідкових зв'язків).

2.2.1 Дерева рішень та їх структура

Для кількісної оцінки та бездоганного моделювання невизначеності у складних середовищах, зокрема SAFe [1], використовуються ентропійні дерева рішень [17—19], які дозволяють встановити можливі варіанти розвитку подій та причино-наслідкові зв'язки. При цьому показники невизначеності будуть позбавлені суб'єктивності.

Ентропійне дерево рішень [17, 18] є графічною моделлю, що ілюструє послідовність рішень та випадкових подій, а також їхні можливі результати. Основними компонентами такого дерева є:

– кореневий вузол: початкова точка аналізу, що зазвичай представляє епік або фічу, невизначеність якої необхідно оцінити;

– вузли рішень: зазвичай позначаються квадратом (□) та відображають точки, де необхідно прийняти вибір між кількома альтернативами, а кожна гілка, що виходить з вузла рішень, представляє одну з доступних альтернатив;

– вузли шансу: позначаються колом (○) та точки, де відбувається випадкова подія, результат якої є невизначеним, виходить кілька гілок, кожна з яких відповідає одному з можливих результатів події та має присвоєну ймовірність настання, сума яких для одного вузла дорівнює одиниці;

– кінцеві вузли: позначаються трикутником (△) або просто кінцем гілки, представляють кінцеві стани або результати конкретного сценарію, кожен кінцевий вузол має асоційоване з ним значення (наприклад, оцінка ризику, бізнес-цінності, або інший показник впливу), що відображає результат шляху від кореня до цього кінцевого вузла;

– гілки: лінії, що з'єднують вузли; гілки, що виходять з вузлів рішень, позначають вибір, тоді як гілки, що виходить з вузлів шансу, позначають можливі результати випадкових подій, і для них вказуються відповідні ймовірності.

2.2.2 Порівняння з іншими методами оцінки невизначеності

З метою обґрунтування доцільності застосування ентропійних дерев рішень виконано порівняльний аналіз із поширеними в Agile-середовищах методами оцінки невизначеності. Встановлено, що традиційні підходи, такі як експертні оцінки (зокрема, Planning Poker) та матриці ризиків, попри простоту використання, характеризуються значним впливом суб'єктивних факторів та обмеженим аналітичним потенціалом. Водночас, альтернативні високоточні методи, зокрема імітаційне моделювання, вирізняються високою ресурсоємністю та складністю інтеграції в ітеративні процеси розробки.

Для більш наочного порівняння, ключові характеристики розглянутих методів та ентропійних дерев рішень представлено у табл. 2.5.

Застосування ентропійних дерев рішень у середовищі Scaled Agile Framework (SAFe) обумовлено можливістю поєднання кількісного аналізу з візуалізацією логіки прийняття рішень. Формалізація показників невизначеності дозволяє вико-

Таблиця 2.5 – Порівняння методів оцінки невизначеності та прийняття рішень

Критерій	Експертні оцінки та планування покеру	Матриці ризиків	Імітаційне моделювання	Ентропійні дерева рішень
Вид оцінки	Якісна, суб'єктивна	Якісна / квазі-кількісна	Кількісна, ймовірнісна	Кількісна, об'єктивна
Прозорість логіки	Низька	Середня	Низька (складні моделі)	Висока (візуальна модель)
Врахування взаємозв'язків	Обмежене	Низьке (ізольовані ризики)	Високе	Високе (через гілки дерева)
Складність імплементації	Низька	Низька	Висока	Середня
Масштабованість	Низька (при великій кількості даних)	Середня	Висока (технічно)	Висока (структурований аналіз)
Обчислювальні ресурси	Низькі	Низькі	Високі	Середні
Інтерпретація результатів	Суб'єктивна, якісна	Суб'єктивна, якісна	Складна (розподіли)	Інтуїтивно зрозум. (шляхи, ентропія)
Фокус	Швидка оцінка зусиль	Виявлення критичних ризиків	Детальний ймовірнісний аналіз	Кількісна оцінка невизначеності та прийняття рішень
Застосування в SAFe	Часткове (оцінка Story Points)	Обмежене (загальний огляд ризиків)	Обмежене (складність інтеграції)	Комплексна підтр. (від портфоліо до команд)

нувати об'єктивне порівняння сценаріїв реалізації, мінімізуючи вплив суб'єктивних експертних оцінок. Графічна інтерпретація моделей забезпечує прозорість причинно-наслідкових зв'язків та архітектурних залежностей, що сприяє ефективній комунікації та узгодженню очікувань стейкхолдерів.

Використання метрики інформаційного приросту дозволяє автоматизувати ідентифікацію ключових факторів, що мають найбільший вплив на зниження ентропії, забезпечуючи перехід до прийняття рішень на основі даних (Data-Driven Decision Making). Завдяки можливості динамічного оновлення параметрів, даний математичний апарат демонструє високу адаптивність до ітеративних процесів планування, характерних для методології Agile, дозволяючи актуалізувати моделі в міру надходження нової інформації.

2.2.3 Концептуальна модель застосування ентропійних дерев

На основі аналізу вищезазначенних методів та їх недоліків запропоновано концептуальну модель використання ентропійних дерев для кількісної оцінки невизначеності в SAFe. Модель побудована на принципах теорії інформації та призначена для інтеграції в існуючі процеси планування. Вона включає три взаємопов'язані рівні аналізу:

а) рівень артефактів (епіків та фіч) – для кожного стратегічного епіка або фічі будується дерево проєктних рішень, де вузли представляють альтернативні технічні чи бізнес рішення, а ентропія Шеннона $H_{\text{арт}}$ обчислюється як міра невизначеності, пов'язаної з цим артефактом;

б) рівень Agile Release Train (ART) – невизначеність окремих артефактів агрегується для формування показника внутрішньої невизначеності всього ART: $b_i = f(H_{\text{арт}_1}, H_{\text{арт}_2}, \dots)$, де f – функція агрегації (наприклад, середнє значення або максимум);

в) рівень програми – взаємозалежності між різними ART в межах програми моделюються системою лінійних рівнянь $(E - A)\vec{x} = \vec{b}$, де $\vec{b}$ – вектор власних невизначеностей ART, а A – матриця взаємовпливів між ними; розв'язок $\vec{x}$ дає загальну невизначеність кожного ART з урахуванням мережевих ефектів.

Кожен рівень моделі відповідає певному рівню деталізації в SAFe: рівень артефактів – для командної роботи та детального планування, рівень ART – для координації в межах програмного інкременту, рівень програми – для стратегічного управління портфелем. Така багаторівнева архітектура дозволяє системі масштабуватися разом з організацією.

Головною перевагою запропонованої моделі є перехід від суб'єктивних якісних оцінок (таких як Story Points) до об'єктивної, кількісної метрики невизначеності, що базується на структурі прийнятих та альтернативних рішень. Ця метрика може бути використана для:

– оптимізації комунікації між стейкхолдерами під час PI Planning шляхом виявлення артефактів з найвищою ентропією;

– проактивного управління ризиками на програмному рівні через аналіз критичних залежностей між ART;

– формування більш обґрунтованих зобов’язань на основі кількісної оцінки невизначеності замість суб’єктивних очікувань;

– відстеження прогресу у зменшенні невизначеності впродовж життєвого циклу проєкту.

Таким чином, концептуальна модель створює теоретичну основу для інформаційної системи, спрямованої на підвищення об’єктивності та ефективності комунікації в масштабованих Agile середовищах. У наступних розділах представлено детальну математичну формалізацію цієї моделі та архітектуру її програмної реалізації.

3 МОДЕЛЮВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

У попередньому розділі представлено концептуальну модель інформаційної системи для аналізу невизначеності в SAFe, яка базується на ентропійних деревах рішень. Ця модель включає три рівні аналізу – від окремих артефактів до мережі взаємозалежних Agile Release Trains. Для практичної реалізації такої системи необхідна формальна математична формалізація кожного рівня, що дозволила перейти від концептуальних положень до конкретних алгоритмів та програмної архітектури.

У цьому розділі представлено детальну математичну модель інформаційної системи, розроблену для кількісної оцінки та оптимізації комунікації між стейкхолдерами в середовищі SAFe. Модель побудована на основі апарату теорії інформації (ентропія Шеннона) та лінійної алгебри (системи лінійних рівнянь).

3.1 Моделювання невизначеності на рівні артефактів

Основним джерелом даних для запропонованої інформаційної системи є артефакти – стратегічні епіки та фічі, що підлягають реалізації в рамках програмного інкременту. Кожен такий артефакт характеризується певною мірою невизначеності, яка виникає через наявність альтернативних шляхів реалізації, неоднозначність технічних рішень та зовнішні фактори ризику. Для формалізації цієї невизначеності використовується апарат ентропійних дерев рішень на основі ентропії Шеннона.

Нехай A – конкретний артефакт (епік або фіча), який потребує оцінки. Цей артефакт може бути реалізований через послідовність n ключових рішень $D_1, D_2, \dots, D_n$. Кожне рішення D_i має множину можливих альтернатив $A_{i1}, A_{i2}, \dots, A_{im_i}$. Вибір конкретної альтернативи визначає подальший шлях реалізації артефакту та впливає на кінцевий результат.

Для побудови формальної моделі введемо такі позначення.

Кожному вузлу шансу $v \in N_{\text{chance}}$ зіставляється дискретний розподіл ймовірностей $P_v = \{p_{v1}, p_{v2}, \dots, p_{vk}\}$, де p_{vj} – ймовірність j-го результату події в цьому вузлі, причому $\sum_{j=1}^{k} p_{vj} = 1$.

Ентропія Шеннона для окремого вузла шансу v обчислюється за формулою:

$$H(v) = -\sum_{j=1}^{k} p_{vj} \log_2 p_{vj} \tag{3.1}$$

Загальна ентропія артефакту A визначається як сума ентропій усіх вузлів шансу в його дереві рішень, зважених на ймовірність досягнення цих вузлів:

$$H(A) = \sum_{v \in N_{\text{chance}}} \pi(v) H(v) \tag{3.2}$$

де $\pi(v)$ — ймовірність того, що в процесі реалізації артефакту буде досягнуто вузол v. Ця ймовірність обчислюється як добуток ймовірностей уздовж шляху від кореня дерева до вузла v.

Для ілюстрації принципу роботи моделі розглянемо приклад побудови дерева рішень для фічі «реалізація авторизації через соціальні мережі». Ключові рішення та їх альтернативи для цієї фічі представлено у вигляді дерева на рис. 3.1.

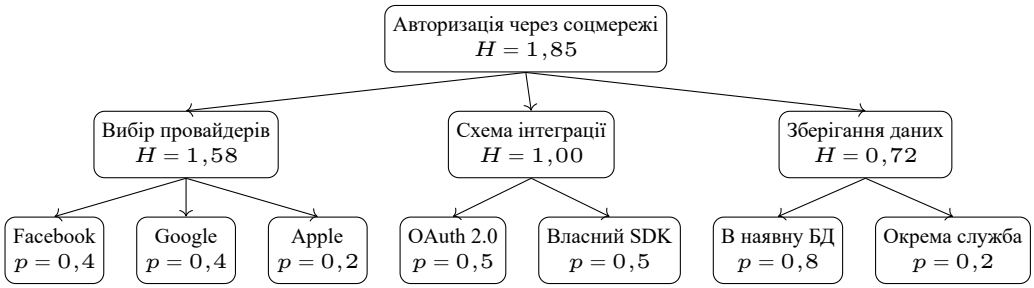

Рисунок 3.1 – Ентропійне дерево рішень для фічі авторизації. Значення ентропії в кожному вузлі обчислені за формулою (3.1)

У цьому прикладі загальна ентропія фічі становить $H(A) = 1{,}85$, що свідчить про відносно високу невизначеність, пов'язану з її реалізацією. Найбільший внесок у цю невизначеність робить вузол вибору провайдерів ($H = 1{,}58$), що вказує на необхідність пріоритетного обговорення цього питання з бізнес стейкхолдерами.

Ймовірності p_{vj} для вузлів шансу можуть бути отримані різними способами:

– експертні оцінки членів команди під час планувальних сесій;
– історичні дані з попередніх подібних проєктів;
– статистичний аналіз частоти вибору тих чи інших альтернатив в організації.

Важливо відзначити, що саме процес побудови дерева рішень та обговорення ймовірностей альтернатив вже є цінним інструментом комунікації, оскільки структурує дискусію та робить приховані припущення явними.

3.2 Агрегація невизначеності на рівні Agile Release Train

В середовищі SAFe Agile Release Train (ART) є основним організаційним підрозділом, який об'єднує кілька команд для доставки цінності в рамках програмного інкременту. Кожен ART відповідає за реалізацію певної множини артефактів – епіків та фіч. Для отримання уявлення про загальну невизначеність, пов'язану з роботою всього ART, необхідно агрегувати показники невизначеності окремих артефактів.

Формально, нехай T_i – i-й Agile Release Train в програмі. Цьому ART призначено множину артефактів $F_i = \{A_1, A_2, \dots, A_k\}$, де k – кількість артефактів у беклозі ART на поточний програмний інкремент. Кожен артефакт A_j має розраховану ентропію $H(A_j)$ згідно з моделлю, представленою в попередньому розділі.

Власна (внутрішня) невизначеність ART T_i позначається як b_i і обчислюється як функція ентропій всіх його артефактів:

$$b_i = f\left(H(A_1), H(A_2), \dots, H(A_k)\right) \tag{3.3}$$

Вибір конкретної функції агрегації f залежить від цілей аналізу та особливостей організації. У запропонованій моделі розглядаються три основні підходи:

а) середня ентропія – $b_i^{\text{avg}} = \frac{1}{k}\sum_{j=1}^{k} H(A_j)$, яка дає уявлення про середній рівень невизначеності артефактів в ART;

б) максимальна ентропія – $b_i^{\max} = \max_{1 \le j \le k} H(A_j)$, яка фокусується на найбільш ризикованому артефакті (принцип «ланцюг рветься по найслабшому ланці»);

в) зважена сума – $b_i^{\text{w}} = \sum_{j=1}^{k} w_j H(A_j)$, де ваги w_j відображають відносну важливість або пріоритет артефактів.

Найпростішим для розуміння та інтерпретації є використання середньої ентропії, проте надалі функція може бути змінена відповідно до потреб.

Розглянемо приклад. Нехай в ART мобільна платформа входить три фічі з такими ентропіями:

A_1 – авторизація через соцмережі: $H(A_1) = 1{,}85$;

A_2 – пуш сповіщення: $H(A_2) = 0{,}92$;

A_3 – офлайн режим: $H(A_3) = 1{,}35$.

Тоді власна невизначеність цього ART обчислюється як:

$$b_{\text{avg}} = \frac{1{,}85 + 0{,}92 + 1{,}35}{3} = 1{,}37,$$
$$b_{\text{max}} = \max(1{,}85; 0{,}92; 1{,}35) = 1{,}85.$$

Різниця між цими двома підходами становить $0{,}48$, що свідчить про значний вплив окремих ризикованих артефактів на загальну оцінку. У практичній реалізації системи передбачена можливість вибору функції агрегації відповідно до потреб конкретної організації.

Вектор $\vec{b} = (b_1, b_2, \dots, b_n)^T$, де n – кількість ART у програмі, становить основу для подальшого аналізу на програмному рівні. Кожен компонент цього вектора відображає внутрішню невизначеність відповідного ART без урахування зовнішніх впливів.

3.3 Мережева модель взаємозалежних Agile Release Train

В умовах масштабованої розробки за методологією SAFindividual Agile Release Trains (ARTs) [1] не є ізольованими одиницями. Вони взаємодіють між собою через технічні залежності, спільні ресурси та узгоджені графіки постачання цінності. Зміни або затримки в одному ART неминуче впливають на інші, створюючи ефект доміно. Для врахування цих взаємозв'язків необхідно перейти від аналізу окремих ART до розгляду всієї програми як єдиної мережі.

Нехай у програмі залучено n Agile Release Trains: $T_1, T_2, \dots, T_n$. Кожен ART T_i має власну невизначеність b_i, обчислену згідно з попереднім розділом. Однак реальна (загальна) невизначеність x_i ART T_i залежить не тільки від його внутрішніх факторів, але й від впливу інших ART.

Для формалізації цих взаємовпливів введемо матрицю $A = [a_{ij}]_{n \times n}$, де коефіцієнт a_{ij} визначає ступінь впливу невизначеності ART T_j на ART T_i. Ці коефіцієнти задовольняють умовам:

$0 \leq a_{ij} \leq 1$ для всіх i, j;

$a_{ii} = 0$ для всіх i (власна невизначеність уже врахована в b_i);

чим тісніша залежність між ART, тим більше значення a_{ij}.

Матриця A є розрідженою, оскільки в реальних програмах кожен ART зазвичай взаємодіє лише з обмеженою кількістю інших ART.

Загальна невизначеність кожного ART описується системою лінійних рівнянь:

$$\begin{cases} x_1 = a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n + b_1, \\ x_2 = a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n + b_2, \\ \vdots \\ x_n = a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n + b_n, \end{cases} \tag{3.4}$$

де a_{ij} — коефіцієнт, який відображає ступінь впливу невизначеності j-го ART на i-й ART;

x_i — загальна невизначеність i-го ART з урахуванням мережевих ефектів;

b_i — вільний член, який характеризує власну, незалежну від інших ART, невизначеність i-го ART.

У матричній формі система (3.4) записується як:

$$\vec{x} = A\vec{x} + \vec{b}, \tag{3.5}$$

де $A = (a_{ij})_{n \times n}$ — матриця взаємних впливів;

$\vec{x} = (x_1, x_2, \dots, x_n)^T$ — вектор невизначеностей;

$\vec{b} = (b_1, b_2, \dots, b_n)^T$ — вектор власних невизначеностей.

Після перетворення отримуємо стандартну форму системи лінійних рівнянь:

$$(E - A)\vec{x} = \vec{b} \tag{3.6}$$

3.3.1 Аналіз сумісності системи залежностей

Для аналізу розв'язності системи рівнянь (3.6) застосовується теорема Кронекера-Капеллі [20]. Ця теорема встановлює необхідну та достатню умову існування розв'язку системи лінійних рівнянь.

Нехай $M = E - A$ – основна матриця системи, а $(M|\vec{b})$ – розширена матриця, отримана дописуванням вектора $\vec{b}$ до матриці M справа. Тоді система лінійних рівнянь $M\vec{x} = \vec{b}$ має розв'язок тоді й тільки тоді, коли ранги цих матриць збігаються:

$$\text{rang}(M) = \text{rang}(M|\vec{b}) \tag{3.7}$$

Інтерпретація умови (3.7) в контексті SAFe має практичне значення для оцінки життєздатності плану програмного інкременту.

Якщо $\text{rang}(M) = \text{rang}(M|\vec{b}) = n$, то система має єдиний розв'язок; це ідеальний випадок, коли мережа залежностей між ART є повною та несуперечливою, і загальну невизначеність кожного ART можна точно обчислити.

Якщо $\text{rang}(M) = \text{rang}(M|\vec{b}) < n$, то система має нескінченну кількість розв'язків; ця ситуація вказує на недостатню визначеність мережі залежностей, наприклад, на відсутність інформації про критичні зв'язки між деякими ART, що потребує уточнення під час PI Planning.

Якщо $\text{rang}(M) \neq \text{rang}(M|\vec{b})$, то система несумісна і не має жодного розв'язку; це сигналізує про наявність суперечливих залежностей у плануванні, коли вимоги до різних ART взаємно виключають одна одну, що є критичною помилкою, яка потребує негайного втручання керівника програми.

На практиці матриця $M = E - A$ зазвичай є добре обумовленою [20], оскільки діагональні елементи $1 - a_{ii} = 1$ домінують над недіагональними. Це забезпечує існування єдиного розв'язку для більшості реальних конфігурацій.

3.3.2 Методика заповнення матриці взаємовпливів

Коефіцієнти a_{ij} матриці A можуть бути отримані кількома способами:

а) експертні оцінки під час PI Planning – учасники оцінюють силу залежності між ART за шкалою від 0 до 1;

б) аналіз архітектурних залежностей – на основі документації по системній архітектурі;

в) історичні дані – аналіз впливу змін в одному ART на інші в минулих програмних інкрементах;

г) автоматичне виявлення – аналіз спільних кодових баз, залежностей між мікросервісами тощо.

Для спрощення початкового впровадження системи використано експертні оцінки під час сесій PI Planning, де представники різних ART можуть спільно оцінити ступінь взаємозалежності.

Розглянемо приклад програми з трьома ART: фронтенд (T_1), бекенд (T_2) та мобільний додаток (T_3). Приклад такої матриці:

$$A = \begin{pmatrix} 0 & 0{,}3 & 0{,}1 \\ 0{,}2 & 0 & 0{,}4 \\ 0{,}05 & 0{,}1 & 0 \end{pmatrix}$$

Це означає, що:

– фроненд (T_1) залежить від бекенду (T_2) з коефіцієнтом $0{,}3$ та від мобільного (T_3) з коефіцієнтом $0{,}1$;

– бекенд (T_2) залежить від фронтенду (T_1) з коефіцієнтом $0{,}2$ та від мобільного (T_3) з коефіцієнтом $0{,}4$;

– мобільний (T_3) залежить від фроненду (T_1) з коефіцієнтом $0{,}05$ та від бекенду (T_2) з коефіцієнтом $0{,}1$.

Припустимо, що власні невизначеності цих ART становлять : $\vec{b} = (1.2, 0.8, 1.5)^T$. Тоді система рівнянь набуває вигляду:

$$\begin{pmatrix} 1 & -0{,}3 & -0{,}1 \\ -0{,}2 & 1 & -0{,}4 \\ -0{,}05 & -0{,}1 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 1{,}2 \\ 0{,}8 \\ 1{,}5 \end{pmatrix}$$

Розв'язок цієї системи дає загальні невизначеності з урахуванням мережевих ефектів.

3.4 Алгоритмічне забезпечення та вибір технологій

Алгоритм роботи інформаційної системи містить такі основні етапи:

а) збір та введення даних – користувачі (продукт оунери, архітектори, керівники ART) будують дерева рішень для артефактів, вказують альтернативи та ймовірності;

б) обчислення ентропій артефактів – для кожного артефакту розраховується ентропія $H(A_j)$ згідно з формулою (3.2);

в) агрегація на рівні ART – для кожного ART обчислюється власна невизначеність b_i згідно з обраною функцією агрегації (3.3);

г) формування матриці взаємовпливів – на основі введених даних про залежності між ART формується матриця A;

д) розв'язання системи рівнянь – розв'язується система $(E - A)\vec{x} = \vec{b}$ для отримання вектору загальних невизначеностей $\vec{x}$;

е) візуалізація результатів – результати представляються у зручному для сприйняття вигляді – графіки залежностей, теплові карти невизначеності, рекомендації щодо пріоритизації робіт.

Для розв'язання систем лінійних рівнянь у запропонованій системі використовується бібліотека Math.NET Numerics [21] для мови C#. Цей вибір обґрунтований такими чинниками:

– висока обчислювальна ефективність – бібліотека містить ретельно оптимізовані алгоритми лінійної алгебри;

– підтримка різних форматів матриць – включаючи розріджені матриці, що важливо для моделювання мереж залежностей між ART;

– надійність – наявність механізмів обробки вироджених та погано обумовлених систем;

– інтеграція з екосистемою .NET – бібліотека легко інтегрується з іншими компонентами системи, розробленими на C#;

– відкритий вихідний код та активна підтримка спільноти.

Бібліотека автоматично вибирає оптимальний алгоритм розв’язання на основі властивостей матриці M: для щільних матриць використовується LU-розклад з частковим вибором опорного елемента, для розріджених – ітераційні методи.

3.5 Програмна реалізація інформаційної системи

У цьому розділі описано особливості програмної реалізації інформаційної системи для оцінки невизначеності в SAFe. Розроблений прототип базується на математичних моделях, обґрунтованих у попередніх розділах, та реалізований як веб-орієнтований застосунок.

Для розробки системи обрано архітектурний патерн MVC (Model-View-Controller) [22], реалізований на базі фреймворку ASP.NET Core [23]. Цей вибір зумовлений необхідністю чіткого розмежування логіки обробки даних, інтерфейсу користувача та керування запитами, що забезпечує гнучкість, тестопридатність та масштабованість системи. Архітектура застосунку складається з трьох основних компонентів, взаємодія яких забезпечує виконання бізнес-логіки системи:

а) модель (Model) – відповідає за представлення даних та бізнес-логіку. У розробленій системі модель складається з двох частин, а саме доменних сутностей (Domain Models), які відображають структуру предметної області (класи `Artifact`, `DecisionTree`, `Node`, `ART`, `DependencyMatrix`), та бізнес-сервісів (Services), що інкапсулюють логіку розрахунків. Для реалізації математичного апарату використовуються сервіси `EntropyService` (розрахунок ентропії Шеннона) та `MatrixSolverService` (розв’язання систем лінійних рівнянь);

б) вигляд (View) – відповідає за відображення даних користувачеві та формування інтерфейсу. Реалізовано за допомогою технології Razor Pages та JavaScript. Основні елементи вигляду включають інтерактивний редактор дерев рішень з підтримкою Drag&Drop, редактор матриці взаємовпливів та аналітичні дашборди для візуалізації графіків та діаграм з використанням бібліотеки Chart.js;

в) контролер (Controller) – обробляє вхідні запити від користувача, звертається до моделі для виконання операцій та вибирає відповідний вигляд для відображення результату. У системі реалізовано основні контролери для управління життєвим циклом епіків та фіч (`ArtifactController`), обробки запитів на створення дерев (`DecisionTreeController`), агрегації даних (`ARTController`) та управління залежностями (`MatrixController`).

Для наочного представлення взаємодії компонентів системи розроблено структурну схему архітектури, яка відображає потік даних від запиту користувача до відповіді сервера (рис. 3.2).

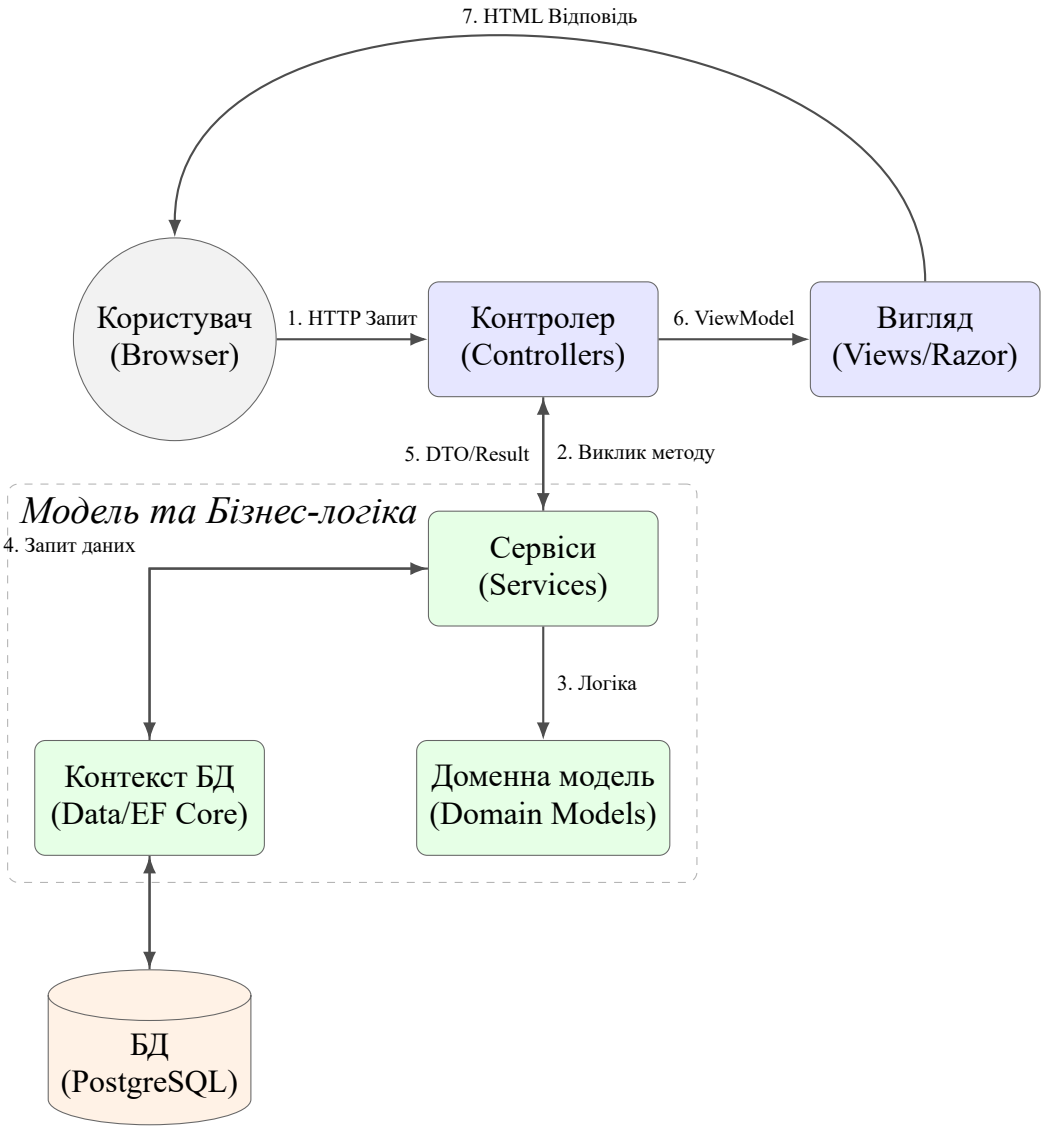

Рисунок 3.2 – Схема архітектури MVC розробленої системи

На рис. 3.2 відображено розподіл відповідальності між компонентами системи:

а) користувач ініціює дію в браузері, надсилаючи HTTP-запит до сервера;

б) контролер приймає запит, валідує вхідні дані та звертається до відповідного сервісу (наприклад, `EntropyService`);

в) сервіс виконує бізнес-логіку, використовуючи доменні моделі;

г) шар доступу до даних (Data) через Entity Framework Core виконує запити до бази даних PostgreSQL для отримання або збереження стану;

д) результати обробки повертаються контролеру;

е) контролер передає дані у Вигляд (View);

ж) сформована HTML-сторінка відображається користувачеві.

Такий поділ дозволив ізолювати складні математичні обчислення, для яких використовується бібліотека `Math.NET Numerics`, від логіки відображення. Фізична структура проєкту відповідає прийнятим стандартам розробки на платформі .NET 8. Основні директорії та файли проєкту включають:

– `/Controllers` – містить класи контролерів API та MVC, що забезпечують маршрутизацію запитів;

– `/Models` – містить описи сутностей бази даних та об’єктів передачі даних (DTO);

– `/Services` – реалізація алгоритмів розрахунку ентропії, агрегації та роботи з матрицями;

– `/Data` – контекст бази даних (Entity Framework Core) та файли міграцій;

– `/wwwroot` – статичні файли фронтенду, зокрема скрипти для візуалізації графів та стилі оформлення.

В якості системи управління базами даних використовується PostgreSQL [24]. Взаємодія з БД реалізована через ORM Entity Framework Core, що забезпечує автоматичне створення та міграцію схеми даних на основі доменних моделей.

3.6 Проєктування структури класів та моделі даних

Розробка інформаційної системи виконувалася з використанням об’єктно-орієнтованого підходу. Ключовим етапом проєктування було створення доменної

моделі, яка відображає сутності предметної області (Agile-процеси) та зв’язки між ними.

3.6.1 Опис реалізованих класів

Центральним елементом архітектури є набір класів, розташованих у просторі імен `SafeEntropySystem.Models.Domain`. Для моделювання ієрархічної структури дерева рішень використано механізм успадкування, де базовий абстрактний клас `Node` визначає спільну поведінку для вузлів рішень та вузлів шансу. Структура основних класів системи та зв’язки між ними наведені на рис. 3.3.

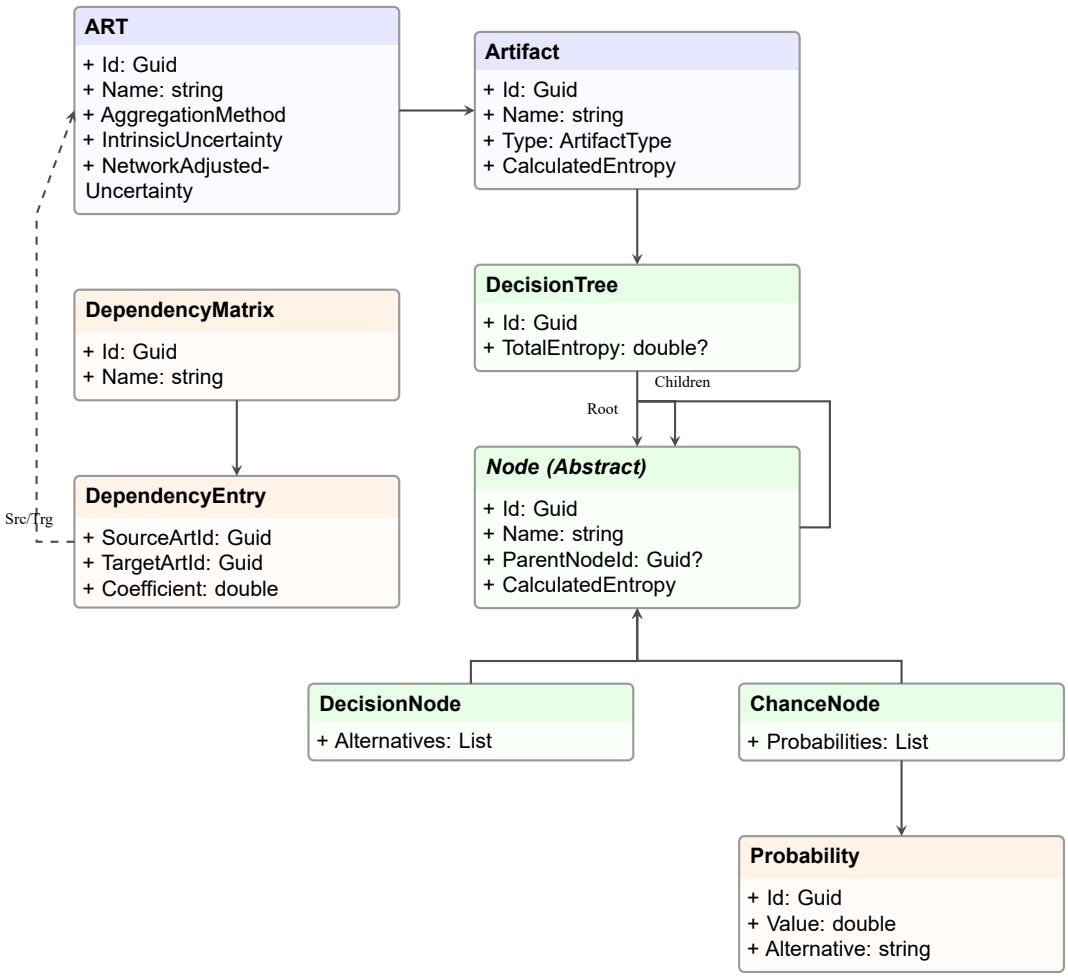

Рисунок 3.3 – Діаграма класів доменної моделі системи

У ході реалізації розроблено набір класів, які повністю описують математичну модель системи:

– `ART` – сутність, що представляє Agile Release Train. Окрім описових полів, містить властивість `AggregationMethod` (метод агрегації ентропії: середнє, максимум або зважена сума) та розраховані поля `IntrinsicUncertainty` (власна невизначеність) і `NetworkAdjustedUncertainty` (мережева невизначеність). Має зв'язок типу один-до-багатьох з артефактами;

– `Artifact` – базовий клас для фіч та епіків (визначається полем `Type`). Містить кешоване значення `CalculatedEntropy` та посилання на корінь дерева рішень `DecisionTreeId`;

– `DecisionTree` – контейнер для структури рішень. Зберігає посилання на кореневий вузол (`RootNode`) та загальну ентропію;

– `Node` – абстрактний базовий клас для вузлів дерева. Реалізує ієрархічну структуру через список `Children` та посилання на батьківський вузол `ParentNode`. Містить координати `PositionX/Y` для візуалізації на фронтенді;

– `DecisionNode` та `ChanceNode` – спадкоємці класу `Node`. Позначає точку прийняття рішення та містить колекцію ймовірностей (`Probabilities`), необхідних для розрахунку ентропії відповідно;

– `DependencyMatrix` та `DependencyEntry` – класи для моделювання мережевих ефектів. Матриця містить набір записів (`Entries`), кожен з яких визначає коефіцієнт впливу a_{ij} одного ART на інший.

Рис. 3.3 містить лише діаграму класів моделі, повна структура проєкту Visual Studio показана на рис. 3.4.

3.6.2 Проєктування бази даних

Збереження стану системи реалізовано на базі реляційної СКБД PostgreSQL. Оскільки розробка велася за підходом Code-First, схема бази даних повністю відповідає об'єктній моделі. Для реалізації ієрархії успадкування класу `Node` використано стратегію TPH (Table Per Hierarchy), де всі типи вузлів зберігаються в одній таблиці з колонкою-дискримінатором.

Модель сутність-зв'язок (ER-діаграма) наведена на рис. 3.5.

База даних містить такі основні таблиці:

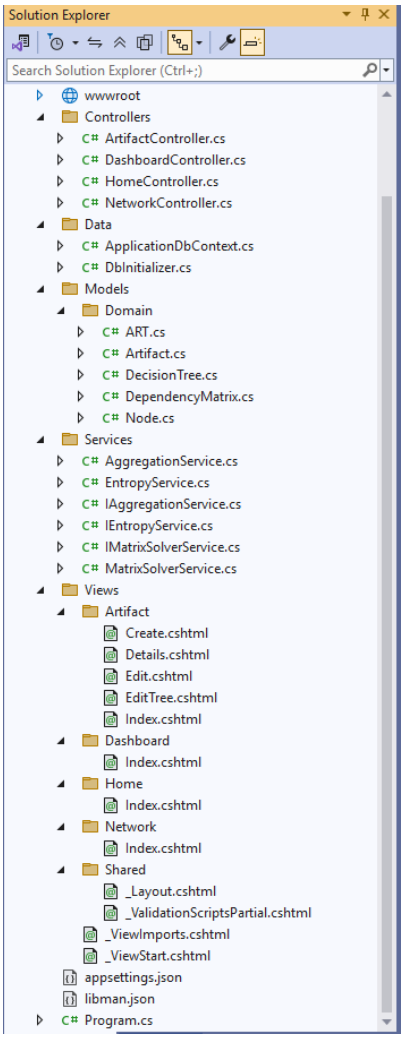

Рисунок 3.4 – Структура проєкту Visual Studio

– `ARTs` – зберігає інформацію про потяги релізів. Містить розраховані поля `IntrinsicUncertainty` для швидкого доступу без необхідності перерахунку;

– `Artifacts` – таблиця епіків та фіч. Має зовнішній ключ `ArtId` для реалізації зв'язку один-до-багатьох з ART;

– `Nodes` – зберігає всі вузли дерев. Завдяки полю `Discriminator` (згідно з патерном TPH) у цій таблиці зберігаються як звичайні вузли, так і вузли шансу. Поле `ParentNodeId` забезпечує рекурсивну структуру дерева;

– `Probabilities` – окрема таблиця для зберігання значень ймовірностей альтернатив, пов'язана з вузлами шансу;

– `DependencyEntries` – таблиця, що реалізує зв'язок багато-до-багатьох між ART, зберігаючи коефіцієнти взаємного впливу.

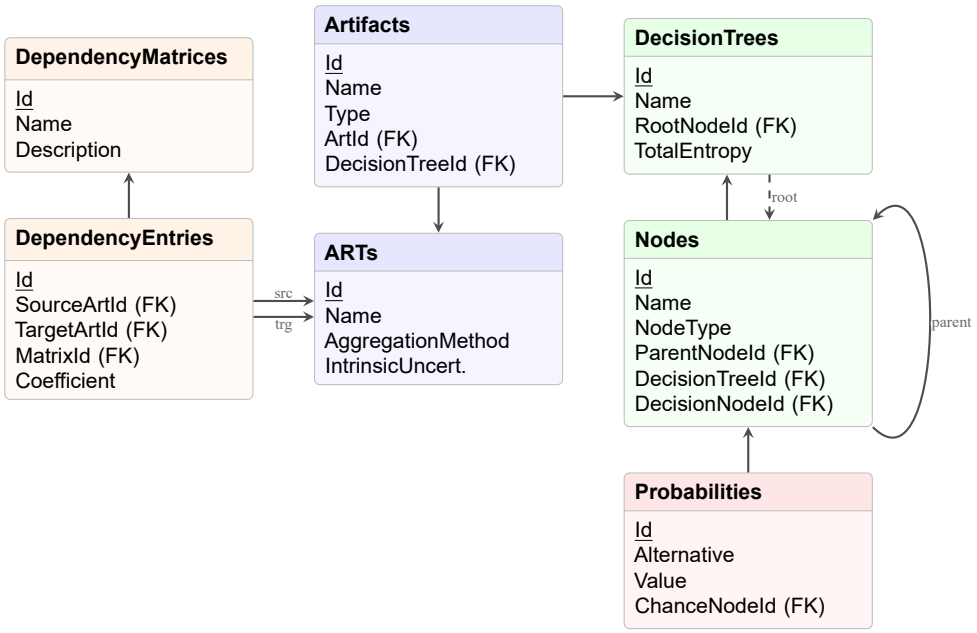

Рисунок 3.5 – Схема бази даних (ER-діаграма)

3.7 Реалізація ключових алгоритмів системи

Основне функціональне навантаження системи зосереджено у шарі сервісів (Services), де імплементовано математичні моделі оцінки невизначеності. Цей шар ізольований від контролерів та бази даних, що дозволяє покрити його модульними тестами та гарантувати коректність розрахунків. Розглянемо детальніше програмну реалізацію двох критичних компонентів: розрахунку ентропії дерев рішень та аналізу мережевих залежностей.

3.7.1 Алгоритм розрахунку ентропії

За обчислення невизначеності відповідає сервіс `EntropyService`. Його основна задача – обійти деревоподібну структуру артефакту, ідентифікувати вузли з імовірнісними подіями та застосувати формулу ентропії Шеннона.

Розрахунок починається з `RecalculateArtifactEntropyAsync`, який виконує підготовчу роботу. Оскільки дерево рішень зберігається у реляційній базі даних у вигляді окремих записів вузлів, пов’язаних зовнішніми ключами, першим кроком є повне завантаження структури дерева в пам’ять. Для цього використовується рекурсивний метод `LoadTreeRecursiveAsync`, який за допомогою

Entity Framework Core жадібно (eagerly) завантажує дочірні елементи та колекції ймовірностей для кожного вузла. Це дозволяє уникнути проблеми «N+1 запитів» та оптимізувати роботу з базою даних.

Безпосередній розрахунок ентропії міститься `CalculateTreeEntropy`. Для оптимізації простору на схемі логіку розбито на паралельні гілки обробки поточного вузла та його нащадків (рис. 3.6).

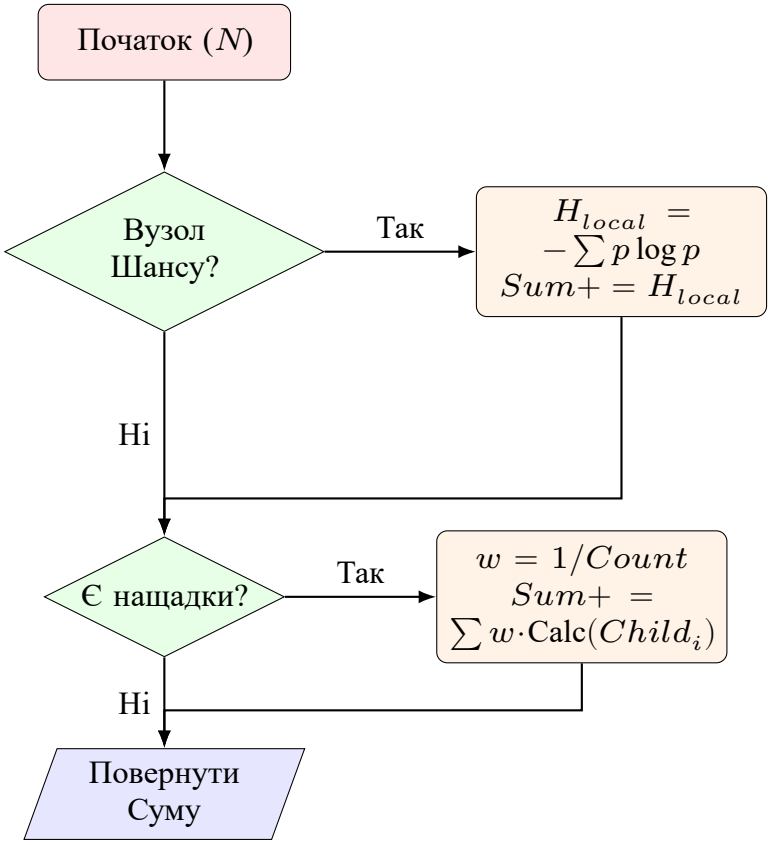

Рисунок 3.6 – Схема алгоритму розрахунку ентропії

Алгоритм містить такі кроки:

а) валідація вхідних даних – система перевіряє, чи існує кореневий вузол. Якщо дерево порожнє, ентропія дорівнює нулю;

б) обробка вузлів шансу (Chance Nodes) – якщо поточний вузол є вузлом шансу, алгоритм аналізує розподіл ймовірностей його альтернатив. Реалізовано критично важливу перевірку: сума ймовірностей усіх альтернатив повинна дорівнювати одиниці. Враховуючи особливості роботи з числами з рухомою комою (тип `double`), перевірка здійснюється з точністю $\epsilon = 0{,}001$. Якщо умова пору-

шена, генерується виняток `InvalidOperationException`, що сигналізує про некоректність моделі;

в) застосування формули Шеннона – для коректних вузлів шансу обчислюється локальна ентропія як сума добутків ймовірності на її двійковий логарифм (зі знаком мінус). Альтернативи з нульовою ймовірністю ігноруються, щоб уникнути математичної помилки обчислення логарифма нуля;

г) рекурсивний спуск – алгоритм переходить до дочірніх вузлів. При цьому враховується вага гілки: якщо вузол має кілька нащадків, внесок кожного з них у загальну ентропію зважується пропорційно (у поточній реалізації використовується рівномірний розподіл ваги $1/N$, де N – кількість нащадків);

д) агрегація результату – результати рекурсивних викликів підсумовуються, формуючи загальне значення ентропії артефакту $H(A)$, яке записується в базу даних для подальшого аналізу.

3.7.2 Реалізація розв'язувача системи лінійних рівнянь

Другим ключовим компонентом системи є сервіс `MatrixSolverService`, який відповідає за моделювання взаємозалежностей на рівні програми. Його мета – перетворити набір розрізнених зв'язків між Agile Release Trains (ART) у єдину математичну модель та знайти вектор реальної невизначеності $\vec{x}$.

Вхідними даними для алгоритму є список ART (кожен з яких має показник власної невизначеності b_i) та список залежностей, де вказано, який ART впливає на інший та з яким коефіцієнтом. Алгоритм представлено у вигляді схеми на рис. 3.7.

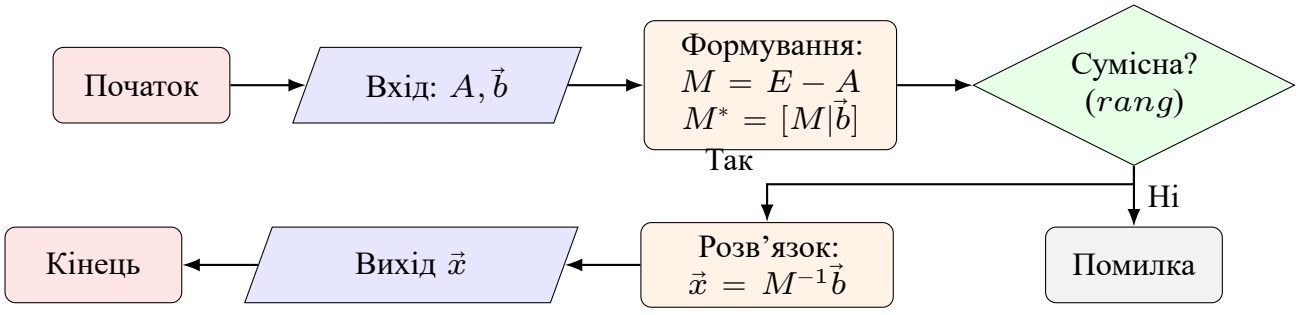

Рисунок 3.7 – Алгоритм розв'язання системи рівнянь

Процес розв'язання складається з кількох етапів:

а) валідація розмірності – перед початком обчислень перевіряється, чи відповідає кількість рівнянь кількості невідомих. Це необхідно для коректної побудови квадратної матриці;

б) побудова матриці системи – формується матриця коефіцієнтів $M = E - A$. Для цього створюється одинична матриця E розмірності $n \times n$, від якої віднімається матриця взаємовпливів A. Елементи на головній діагоналі отримують значення $(1 - a_{ii})$, а решта елементів дорівнюють $-a_{ij}$. Для роботи з матрицями використано типи `Matrix<double>` та `Vector<double>` з бібліотеки Math.NET Numerics;

в) перевірка на сумісність – це критичний етап алгоритму, що запобігає спробам розв'язати некоректно сформульовану систему. Реалізовано метод `CheckConsistency`, який обчислює ранг основної матриці M та ранг розширеної матриці $[M|\vec{b}]$. Згідно з теоремою Кронекера-Капеллі, якщо ранги не збігаються, система не має розв'язку. У такому випадку користувач отримує повідомлення про помилку з рекомендацією переглянути залежності;

г) обчислення розв'язку – якщо система є сумісною та визначеною (матриця невироджена), виконується знаходження оберненої матриці M^{-1} або застосовується метод LU-розкладу. Отриманий вектор $\vec{x}$ містить скориговані значення невизначеності для кожного ART;

д) інтерпретація результатів – числовий вектор перетворюється назад у словник, де ключем є ідентифікатор ART, а значенням – його розрахована мережева невизначеність.

Для моніторингу продуктивності у метод вбудовано вимірювання часу виконання за допомогою `Stopwatch`, що дозволяє оцінити ефективність алгоритму при збільшенні кількості вузлів у мережі.

4 АНАЛІЗ ОТРИМАНИХ РЕЗУЛЬТАТІВ

У цьому розділі продемонстровано результати роботи розробленого прототипа інформаційної системи. Перевірка коректності реалізованих алгоритмів та зручності інтерфейсу здійснювалася на основі тестового набору даних, що відтворює типову структуру залежностей у програмному проєкті.

Нижче наведено послідовний опис роботи з системою, починаючи з огляду загального стану проєкту та ключових показників.

Взаємодія користувача з системою починається з головної сторінки, яка виконує роль центрального хабу для навігації та моніторингу стану проєкту (рис. 4.1).

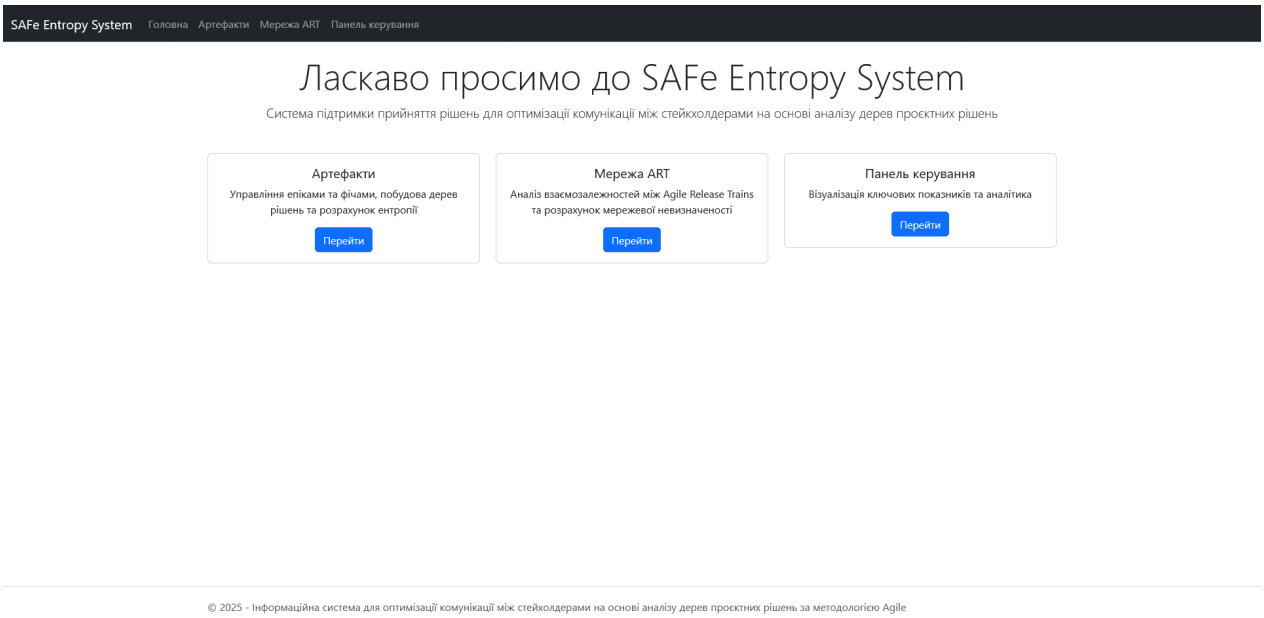

Рисунок 4.1 – Головна сторінка системи

Сторінка розділена на три функціональні блоки:

а) артефакти – модуль для управління беклогом епіків та фіч. Тут користувач може створювати нові задачі, прив’язувати їх до конкретних Agile Release Trains (ART) та переходити до побудови дерев рішень;

б) мережа ART – модуль для моделювання залежностей на рівні програми. Забезпечує доступ до матриці взаємовпливів та запуску алгоритму розрахунку мережевої невизначеності;

в) панель керування – аналітичний блок для візуалізації ключових метрик.

Така структура інтерфейсу дозволяє розділити процеси введення даних (рівень команди) та аналізу ризиків (рівень програми), що відповідає ролям Product Owner та Release Train Engineer у методології SAFe.

Центральним елементом роботи з вимогами в системі є реєстр артефактів, доступний за маршрутом `/Artifact`. Ця сторінка містить всі стратегічні задачі (епіки та фічі) та надає зведену інформацію про їхній стан (рис. 4.2).

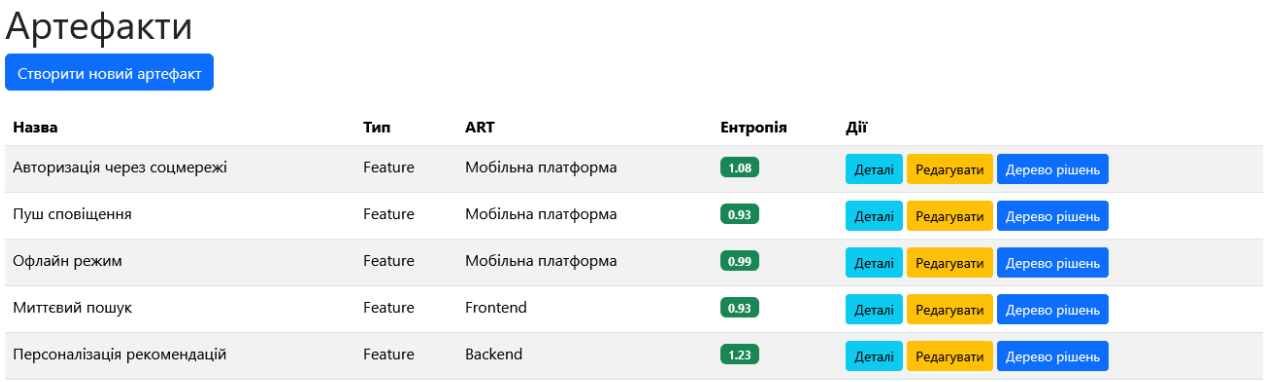

Рисунок 4.2 – Інтерфейс реєстру артефактів з індикацією рівня ентропії

Інтерфейс реєстру реалізовано у вигляді таблиці, що містить такі ключові атрибути:

– назва артефакту, яка ідентифікує задачу в системі та тип сутності;

– приналежність до конкретного Agile Release Train, що дозволяє фільтрувати задачі по командах;

– розрахований показник ентропії з кольоровою індикацією рівня ризику.

Система автоматично виконує категоризацію ризиків на основі обчисленого значення ентропії: зелений колір позначає низьку невизначеність, жовтий – середню, а червоний сигналізує про високий ризик. Наприклад, як видно з рисунка, фіча авторизація через соцмережі має показник 1,08 (низький ризик), тоді як персоналізація рекомендацій має вищий показник 1,23, що відображає більшу складність реалізації.

Функціональність модуля передбачає можливість створення нових артефактів через спеціальну форму. При реєстрації нового запису користувач зобов’язаний вказати назву та тип, а також опціонально додати опис та призначити відповідальний ART. Важливо зазначити, що створення запису артефакту є лише етапом

ініціалізації; моделювання структури невизначеності відбувається окремо через редактор дерева рішень, доступ до якого відкривається натисканням відповідної кнопки у списку.

У процесі життєвого циклу задачі може виникнути потреба в актуалізації її параметрів. Для цього в системі реалізовано режим редагування, інтерфейс якого представлено на рис. 4.3.

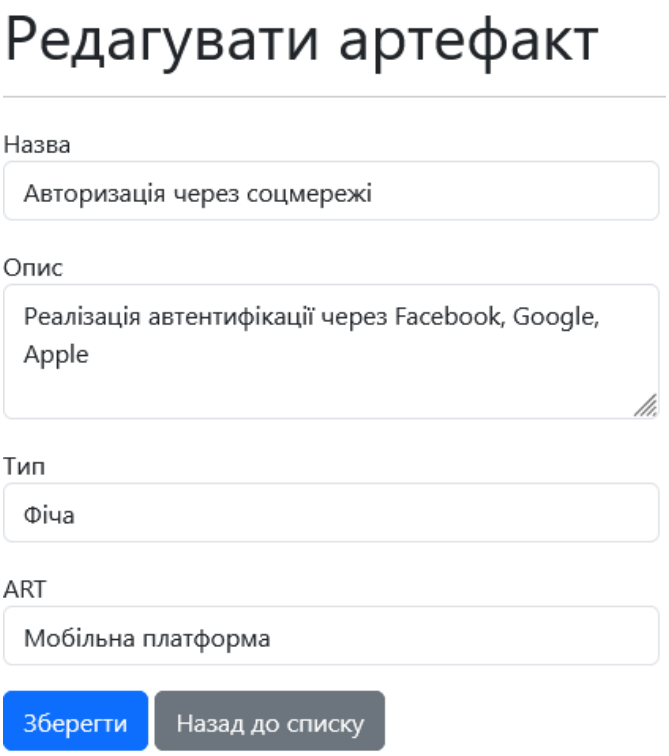

Рисунок 4.3 – Форма редагування параметрів артефакту

Форма редагування дозволяє змінювати назву, текстовий опис, тип артефакту та його приналежність до Agile Release Train. Важливою архітектурною особливістю є те, що зміна цих метаданих не впливає на внутрішню структуру пов’язаного дерева рішень, що забезпечує цілісність даних при рефакторингу вимог.

Для отримання повної інформації про об’єкт без переходу в режим редагування передбачено сторінку деталізації (рис. 4.4). Цей інтерфейс агрегує всі дані про артефакт, включаючи його поточний статус та розрахований показник ентропії.

Візуальний індикатор ентропії дозволяє миттєво оцінити рівень ризику:

– зелений колір свідчить про низьку невизначеність;

– жовтий колір вказує на середній рівень ризику;

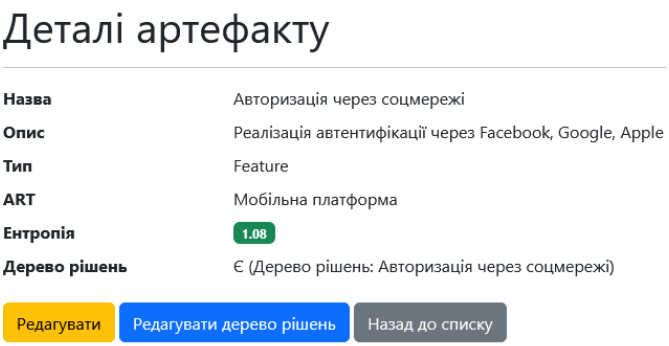

Рисунок 4.4 – Сторінка детальної інформації про артефакт

– червоний колір сигналізує про високу невизначеність, що потребує уваги.

Зі сторінки деталізації користувач має доступ до навігаційних елементів для переходу до моделювання структури рішень або повернення до загального списку.

Ключовим аналітичним інструментом системи є візуальний редактор дерев рішень (рис. 4.5), доступ до якого здійснюється через кнопку дерево рішень у списку артефактів. Цей модуль дозволяє декомпозувати складну задачу на послідовність рішень та ймовірнісних подій.

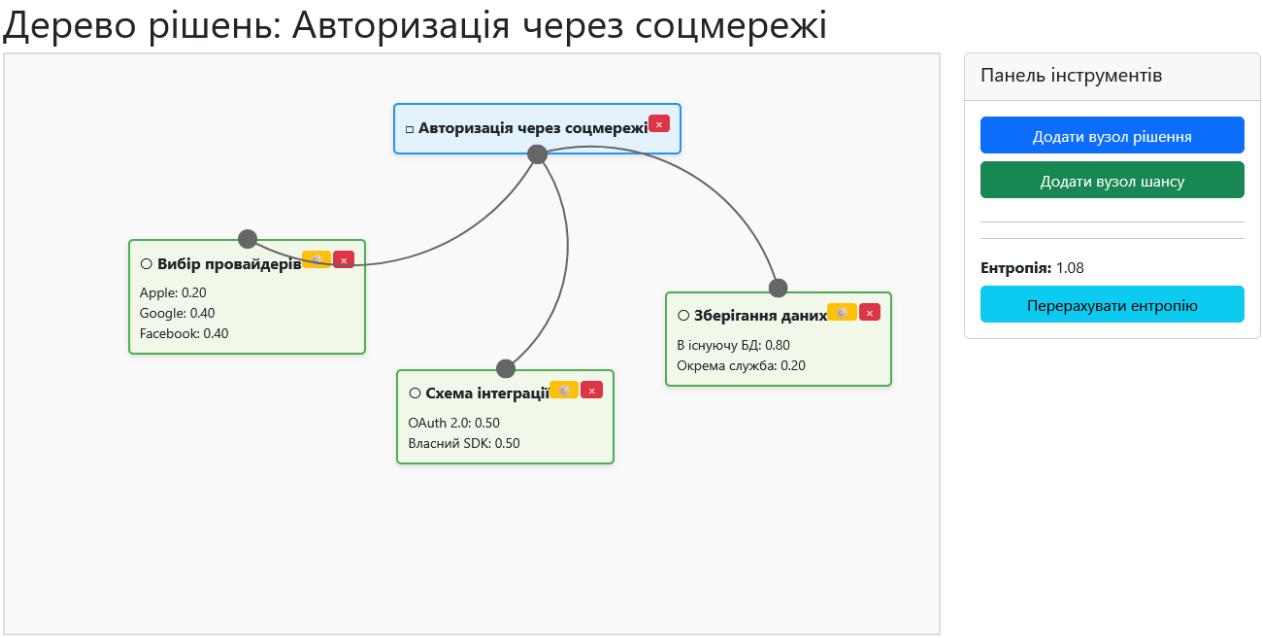

Рисунок 4.5 – Візуальний редактор дерева рішень для фічі авторизації

Робоча область редактора розділена на дві частини:

– ліва частина (Canvas) відображає граф рішень, де вузли з’єднані дугами, що візуалізують потік виконання задачі;

– права частина (панель інструментів) містить кнопки для додавання нових вузлів та відображає поточне значення загальної ентропії артефакту.

У системі використовуються два типи вузлів:

– вузол рішення (позначається синім кольором) – точка, де команда має свідомо обрати один із варіантів реалізації;

– вузол шансу (позначається зеленим кольором) – точка невизначеності, де результат залежить від зовнішніх факторів або технічних ризиків. Кожна гілка, що виходить з такого вузла, має визначену ймовірність.

На наведеному рисунку показано модель реалізації фічі авторизації через соцмережі. Кореневий вузол рішення розгалужується на три підзадачі (вузли шансу): вибір провайдерів, схема інтеграції та зберігання даних. Для кожної альтернативи (наприклад, Facebook, Google, Apple) визначено ймовірність вибору. Система автоматично агрегує ці дані, розраховуючи сумарну ентропію $H = 1{,}08$, що відображається на панелі інструментів.

Функція перерахунку ентропії дозволяє миттєво оцінити вплив зміни ймовірностей на загальну невизначеність, що робить цей інструмент ефективним засобом для моделювання сценаріїв що-якщо.

Процес детального аналізу артефактів відбувається шляхом декомпозиції задачі на окремі кроки. Інструментарій редактора дозволяє формувати ієрархічну структуру, починаючи зі створення кореневого елемента, який автоматично розміщується в центрі робочого полотна.

Подальша розбудова дерева здійснюється шляхом додавання дочірніх вузлів. При виборі батьківського елемента користувач має можливість додати новий вузол рішення або вузол шансу, при цьому система автоматично встановлює зв’язки та візуалізує їх у вигляді дуг. Інтерфейс підтримує функцію перетягування (drag-and-drop), що дозволяє оптимізувати візуальне представлення графа для кращого сприйняття.

Для кожного вузла доступні функції редагування назви та зміни ієрархічної підпорядкованості, що активуються через панель інструментів при виділенні елемента (рис. 4.6).

Рисунок 4.6 – Процес редагування структури та параметрів вузлів

Окремої уваги потребує налаштування стохастичних параметрів моделі. Вузли шансу (зеленого кольору) містять спеціальні елементи керування для переходу до налаштування розподілу ймовірностей (рис. 4.7).

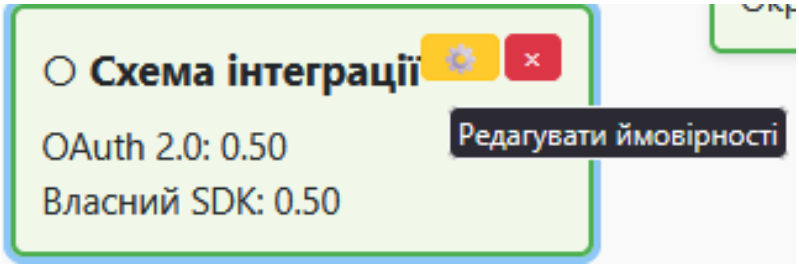

Рисунок 4.7 – Елементи управління вузлом шансу

Натискання на піктограму налаштувань відкриває модальне вікно (рис. 4.8), де користувач визначає перелік можливих наслідків події та присвоює кожному з них числове значення ймовірності. Система містить вбудований механізм валідації, який слідкує за тим, щоб сума ймовірностей усіх альтернатив суворо дорівнювала одиниці.

Будь-які зміни в топології дерева або значеннях ймовірностей ініціюють автоматичний перерахунок ентропії. Це забезпечує інтерактивність процесу моделювання та дозволяє миттєво бачити вплив локальних змін на загальну оцінку невизначеності артефакту.

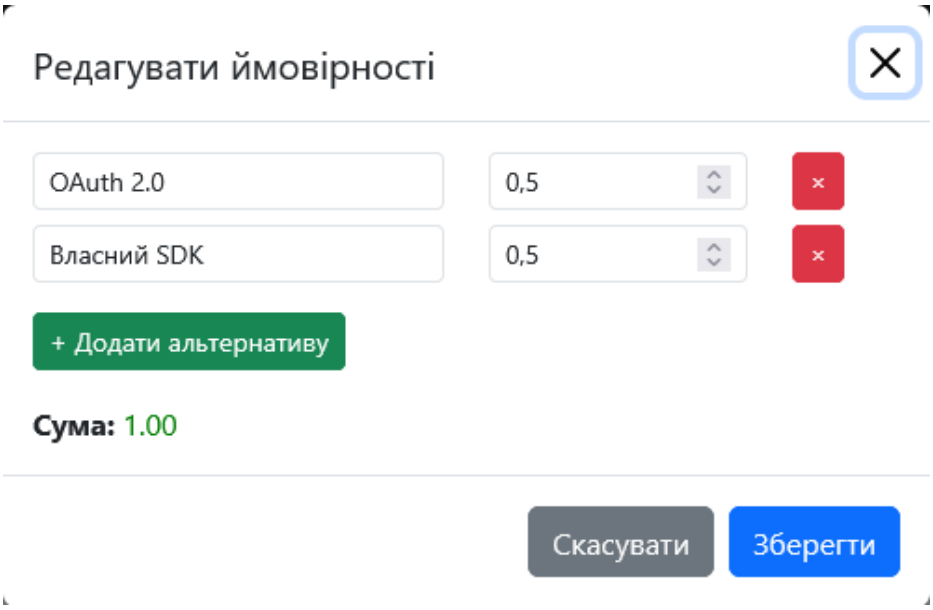

Рисунок 4.8 – Вікно налаштування розподілу ймовірностей альтернатив

Під час роботи з редактором система забезпечує автоматичний контроль цілісності даних та дотримання математичних обмежень моделі. Зокрема, реалізовано низку важливих правил:

– автоматична ініціалізація структури дерева відбувається при створенні першого вузла, який стає коренем;

– для нових вузлів шансу система за замовчуванням створює дві альтернативи з рівномірним розподілом ймовірностей (0,5 кожна);

– перед збереженням змін виконується обов'язкова перевірка умови нормування: сума ймовірностей усіх альтернатив у вузлі шансу повинна дорівнювати одиниці;

– видалення вузла реалізовано за каскадним принципом, що призводить до видалення всієї підпорядкованої гілки, запобігаючи порушенню зв'язності графа;

– будь-яка модифікація топології або числових параметрів ініціює автоматичний перерахунок загальної ентропії артефакту.

Розроблений інструментарій підтримує декілька типових сценаріїв використання, що покривають потреби різних етапів життєвого циклу розробки.

Сценарій 1: проста оцінка альтернатив. Використовується для швидкого аналізу локальних технічних рішень (наприклад, вибір технологічного стеку). Користувач створює один вузол рішення та додає до нього набір альтернатив через дочірні

вузли шансу. Результатом є миттєва оцінка невизначеності вибору без побудови глибокої ієрархії.

Сценарій 2: багаторівневе моделювання. Застосовується для складних епіків, де наслідки рішень розгалужуються в часі. Процес передбачає ітеративну побудову дерева: для кожного результату вузла шансу можуть бути додані нові дочірні вузли рішень. Це дозволяє врахувати відкладені ризики та залежності.

Сценарій 3: рефакторинг моделі. Цей сценарій є актуальним при зміні вимог або уточненні вхідних даних. Функціонал редактора дозволяє змінювати назви вузлів, коригувати розподіл ймовірностей на основі нової інформації, а також змінювати ієрархію шляхом перепризначення батьківських вузлів, що дає змогу реорганізувати структуру дерева без необхідності його створення з нуля.

Після оцінки окремих артефактів система дозволяє перейти на рівень програми для аналізу взаємозалежностей між командами. Модуль роботи з мережею ART (рис. 4.9) реалізує математичну модель розповсюдження ризиків через матрицю впливу.

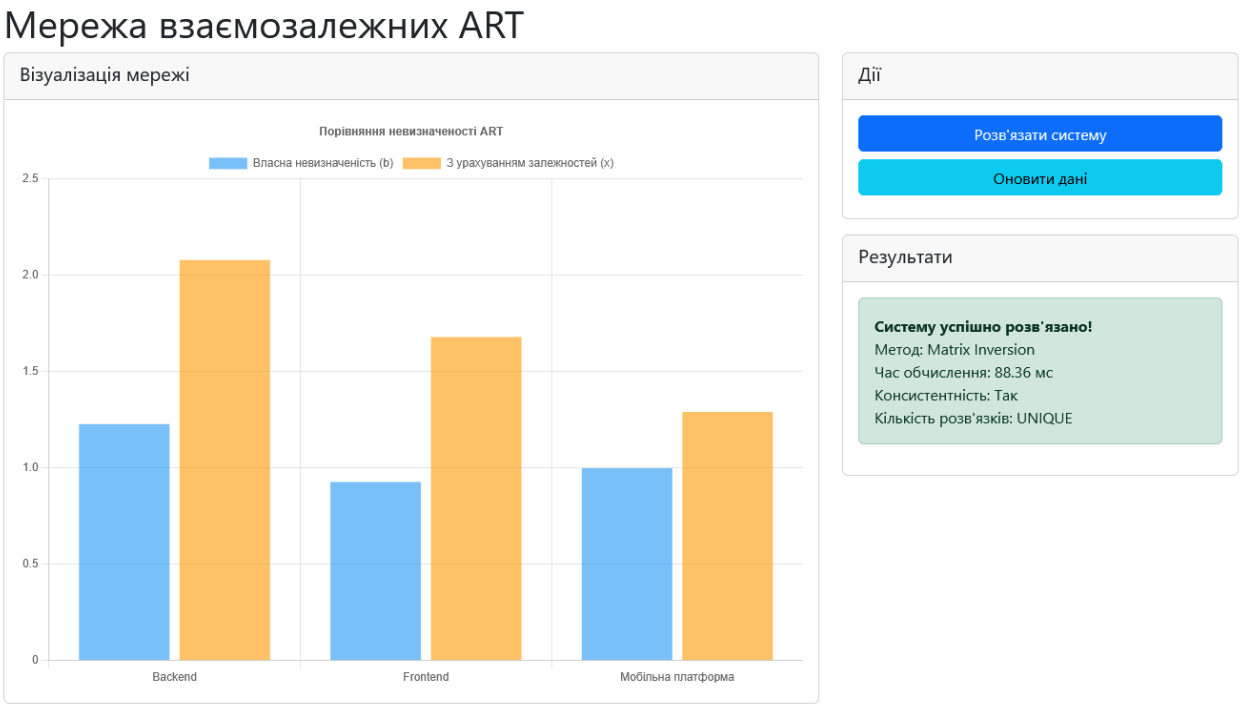

Рисунок 4.9 – Інтерфейс аналізу мережевої невизначеності

Сторінка аналізу розділена на дві функціональні зони:

– область візуалізації (зліва), де у вигляді стовпчастої діаграми відображається порівняння власної та мережевої невизначеності для кожного потягу релізів;

– панель керування (справа), що містить інструменти для запуску розрахунку та блок діагностичної інформації.

Процес аналізу ініціюється натисканням кнопки розв'язати систему. У цей момент серверна частина виконує алгоритм обернення матриці $(E-A)$ та знаходить вектор x. Результати обчислень, включаючи статус консистентності системи та час виконання операції (у наведеному прикладі – 88,36 мс), виводяться у блоці результатів.

Детальні числові показники представлені у табличному вигляді (рис. 4.10). Таблиця містить три ключові метрики для кожного ART:

– власна невизначеність (b) – базовий рівень ризику, розрахований виключно на основі ентропії призначених артефактів;

– мережева невизначеність (x) – скоригований рівень ризику, що враховує вплив інших команд;

– збільшення – відсотковий показник, що демонструє ступінь залежності команди від зовнішніх факторів.

ART та їх невизначеність

ART	Власна невизначеність (b)	Мережева невизначеність (x)	Збільшення
Backend	1.23	2.08	69.4%
Frontend	0.93	1.68	81.2%
Мобільна платформа	1.00	1.29	29.2%

Рисунок 4.10 – Таблиця результатів розрахунку системних ризиків

Наведені на рисунках дані дозволяють зробити важливі управлінські висновки, які неможливо отримати при аналізі ізольованих команд.

Зокрема, привертає увагу ситуація з командою Frontend. Її власна невизначеність є найнижчою серед усіх ($b = 0{,}93$), що може створити хибне враження стабільності. Однак, після врахування мережевих залежностей, показник зростає до 1,68, що становить приріст у 81,2%. Це свідчить про те, що успіх фронтенд-команди критично залежить від стабільності інших підсистем (передусім бекенду).

Такий аналіз дає менеджменту програми можливість:

– ідентифікувати приховані ризики, які не видно на рівні окремих беклогів;

– прогнозувати ланцюгову реакцію затримок ще до початку реалізації програмного інкременту.

Централізований огляд стану системи та візуалізація аналітичних даних реалізовані на сторінці панелі керування (рис. 4.11). Цей екран автоматично агрегує результати розрахунків з усіх модулів та надає менеджеру програми миттєвий зріз поточної ситуації.

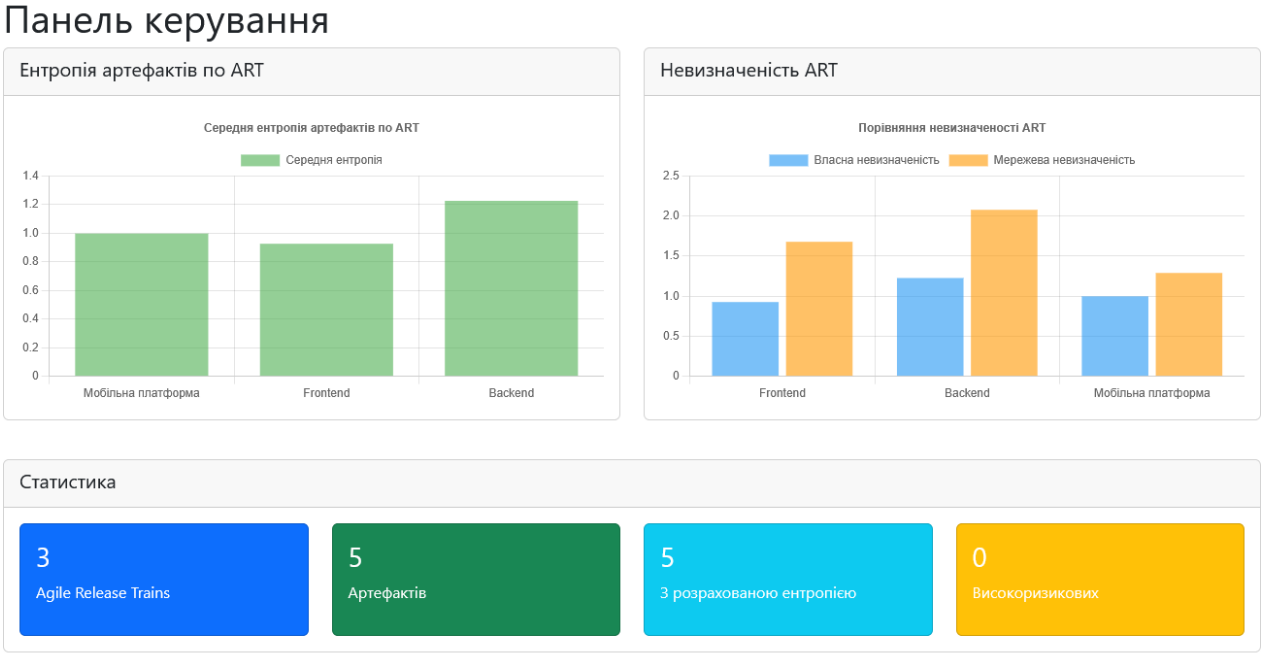

Рисунок 4.11 – Панель керування з аналітичними графіками та статистикою

Верхня частина інтерфейсу містить графічні діаграми для порівняльного аналізу:

– графік середньої ентропії артефактів дозволяє оцінити складність беклогу кожної команди та виявити найбільш навантажені потяги релізів;

– діаграма невизначеності візуалізує різницю між власною (синій колір) та мережевою (помаранчевий колір) ентропією, наочно демонструючи вплив системних залежностей, розрахований на попередньому етапі.

У нижній частині сторінки розміщено блок зведеної статистики, який відображає кількісні метрики проєкту. Така організація дашборда дозволяє стейкхолдерам швидко ідентифікувати проблемні зони без необхідності заглиблюватися в детальні налаштування окремих задач.

ВИСНОВКИ

У кваліфікаційній роботі магістра розв'язано актуальне науково-практичне завдання підвищення ефективності управління проєктами в масштабованих Agile-середовищах. Головним результатом роботи стала розробка та впровадження інформаційної системи, яка дозволяє об'єктивізувати оцінку ризиків та невизначеності програмних артефактів.

На основі аналізу предметної області встановлено, що традиційні методи оцінки, такі як Story Points та Use Case Points, значною мірою спираються на експертні судження. Це призводить до високої суб'єктивності та похибок у плануванні, які стають критичними при масштабуванні процесів у межах фреймворку SAFe. Для подолання цих недоліків було обґрунтовано доцільність застосування ентропійного підходу, що дозволило перейти від інтуїтивних здогадок до математично вивірених показників.

Для реалізації цього підходу розроблено математичну модель, яка базується на апараті теорії інформації та лінійної алгебри. Невизначеність окремих задач (епіків та фіч) моделюється за допомогою ентропійних дерев рішень, де кожен вузол відображає точку вибору або ймовірнісну подію. Для врахування системних ризиків на рівні програми запропоновано метод розрахунку мережевої невизначеності через розв'язання системи лінійних рівнянь вигляду $(E - A)\vec{x} = \vec{b}$. Це дозволило кількісно оцінити вплив взаємозалежностей між командами Agile Release Trains на загальну стабільність плану.

Практичним результатом роботи стала спроєктована та реалізована інформаційна система у вигляді вебзастосунку на платформі .NET 8 із використанням архітектурного патерну MVC, вихідний код якої доступний у репозиторії [25]. Система включає інтерактивний редактор дерев рішень, модуль управління залежностями та аналітичну панель, що забезпечує автоматизацію складних розрахунків та візуалізацію результатів у зручному для сприйняття вигляді.

Експериментальна перевірка розробленого рішення на модельному проєкті підтвердила його ефективність. Результати моделювання показали, що врахування мережевих ефектів дозволяє виявити приховані ризики, які залишаються непомі-

ченими при ізольованому аналізі беклогів команд. Зокрема, у тестовому сценарії виявлено зростання показника невизначеності однієї з команд на 81,2% внаслідок впливу суміжних підсистем, що дозволило вчасно скоригувати план розробки.

Впровадження запропонованої системи дозволяє менеджерам програм та власникам продуктів отримувати об’єктивні дані для прийняття рішень, оптимізувати комунікацію між стейкхолдерами та підвищити передбачуваність виконання програмних інкрементів у середовищі SAFe.

ПЕРЕЛІК ПОСИЛАНЬ

1. Scaled Agile, Inc. Scaled Agile Framework (SAFe). URL: https://www.scaledagileframework.com/ (дата зверн. 21.05.2025).

2. Великодний С. С. Моделі та методи проактивного управління проєктами із розвитку програмних систем і продуктів: монографія. Одеса: Одеський державний екологічний університет, 2021. С. 322. ISBN 978-966-186-182-3.

3. Velykodniy S. S., Burlachenko Z. V., Zaitseva-Velykodna S. S. Rozrobka arkhitektury prohramnoho zasobu dlia upravlinnia merezhevym planuvanniam reinzhynirynhu prohramnoho projektu [Development of software architecture for managing network planning of software project reengineering] // Suchasnyi stan naukovykh doslidzhen ta tekhnolohii v promyslovosti. 2019. 2 (8). С. 25—35. DOI: 10.30837/2522-9818.2019.8.025.

4. Великодний С. С., Тимофєєва О. С. Реінжиніринг програмного забезпечення інформаційних систем: монографія. Одеса: ОДЕКУ, 2020. С. 160.

5. Великодний С. С. Моделювання складних процесів та систем (Частина 1): конспект лекцій. Одеса: Одеський державний екологічний університет, 2021. С. 92. ISBN 978-966-186-181-6.

6. Великодний С. С., Тимофєєва О. С., Зайцева-Великодна С. С. Метод розрахунку показників оцінки проекту при виконанні реінжинірингу програмних систем // Радіоелектроніка, інформатика, управління. 2018. № 4. С. 135—142. DOI: 10.15588/1607-3274-2018-4-13.

7. Asana T. What is IT project management? URL: https://asana.com/resources/it-project-management (дата зверн. 26.11.2024).

8. What is agile project management? URL: https://www.atlassian.com/agile/project-management (дата зверн. 27.11.2024).

9. Wisdom L. Managing Stakeholders in Agile Projects: Mastering Effective Engagement. URL: https://www.leanwisdom.com/blog/managing-stakeholders-in-agile-projects-mastering-effective-engagement (дата зверн. 29.11.2024).

10. Scrum Board Basics: Getting Started with Agile. URL: https://www.atlassian.com/agile/project-management/scrum-board (дата зверн. 27.11.2024).

11. The online Scrum board tool made for team innovation. URL: https://miro.com/agile/scrum-board/ (дата зверн. 27.11.2024).

12. Issue boards. URL: https://docs.gitlab.com/ee/user/project/issue_board.html (дата зверн. 28.11.2024).

13. A Kanban tool for visual planning. URL: https://miro.com/agile/kanban/ (дата зверн. 28.11.2024).

14. Banerjee G. Use Case Points: An Estimation Approach : White Paper / Wipro Technologies. 2001.

15. Radigan D. Story points and estimation. URL: https://www.atlassian.com/agile/project-management/estimation (дата зверн. 07.12.2024).

16. Online planning poker for more inclusive sprints. URL: https://miro.com/agile/planning-poker/ (дата зверн. 09.12.2024).

17. Shannon C. E. A Mathematical Theory of Communication // Bell System Technical Journal. 1948. Т. 27, № 3. С. 379—423.

18. Кулик А. Я., Кривогубченко С. Г. Теорія інформації і кодування: навч. посіб. Вінницький національний технічний університет, 2008.

19. What is a decision tree? URL: https://www.ibm.com/think/topics/decision-trees (дата зверн. 10.04.2025).

20. Клепко В. Ю., Голець В. Л. Вища математика в прикладах і задачах. 2-ге видання. Київ : Центр учбової літератури, 2009. С. 594.

21. Math.NET Numerics. URL: https://numerics.mathdotnet.com/ (дата зверн. 11.11.2025).

22. Fowler M. Patterns of Enterprise Application Architecture. Addison-Wesley Professional, 2002. ISBN 978-0321127426.

23. Lock A. ASP.NET Core in Action. 2nd. Manning Publications, 2020. ISBN 978-1617298301.

24. The PostgreSQL Global Development Group. PostgreSQL 16.2 Documentation. URL: https://www.postgresql.org/docs/16/index.html (дата зверн. 13.11.2025).

25. SafeEntropySystem: Програмна реалізація системи оцінки ентропії в SAFe. URL: https://github.com/andrey-lp/SafeEntropySystem.git (дата зверн. 08.12.2025).

ДОДАТОК А
ПРОГРАМНІ КОДИ МОДЕЛЕЙ ПРОТОТИПУ ІНФОРМАЦІЙНОЇ СИСТЕМИ

У даному додатку наведено фрагменти вихідного коду ключових компонентів розробленої інформаційної системи, які відповідають за практичну реалізацію математичних моделей оцінки невизначеності.

```
public class ART
{
    public Guid Id { get; set; } = Guid.NewGuid();
    public string Name { get; set; } = string.Empty;
    public string Description { get; set; } = string.Empty;
    public DateTime CreatedAt { get; set; } = DateTime.UtcNow;

    // Artifacts assigned to this ART
    public List<Artifact> Artifacts { get; set; } = new();

    // Aggregation method
    public AggregationMethod AggregationMethod { get; set; } =
     ↪  AggregationMethod.Average;

    // Calculated intrinsic uncertainty (b_i)
    public double? IntrinsicUncertainty { get; set; }

    // Network-adjusted uncertainty (x_i) - calculated from matrix
    public double? NetworkAdjustedUncertainty { get; set; }
}

public enum AggregationMethod
{
    Average,
    Max,
    WeightedSum
}

public class Artifact
{
    public Guid Id { get; set; } = Guid.NewGuid();
    public string Name { get; set; } = string.Empty;
    public string Description { get; set; } = string.Empty;
    public ArtifactType Type { get; set; }
```

```
    public DateTime CreatedAt { get; set; } = DateTime.UtcNow;
    public DateTime? UpdatedAt { get; set; }

    // Navigation
    public Guid? DecisionTreeId { get; set; }
    public DecisionTree? DecisionTree { get; set; }

    public Guid? ArtId { get; set; }
    public ART? Art { get; set; }

    // Calculated entropy (cached)
    public double? CalculatedEntropy { get; set; }
}

public enum ArtifactType
{
    Feature,
    Epic
}

public class DecisionTree
{
    public Guid Id { get; set; } = Guid.NewGuid();
    public string Name { get; set; } = string.Empty;
    public DateTime CreatedAt { get; set; } = DateTime.UtcNow;

    // Root node
    public Guid? RootNodeId { get; set; }
    public Node? RootNode { get; set; }

    // Calculated total entropy
    public double? TotalEntropy { get; set; }
}

public class DependencyMatrix
{
    public Guid Id { get; set; } = Guid.NewGuid();
    public string Name { get; set; } = string.Empty;
    public string? Description { get; set; }
    public DateTime CreatedAt { get; set; } = DateTime.UtcNow;

    // Matrix entries
    public List<DependencyEntry> Entries { get; set; } = new();
}
```

```
public class DependencyEntry
{
    public Guid Id { get; set; } = Guid.NewGuid();

    // Source ART (i)
    public Guid SourceArtId { get; set; }
    public ART? SourceArt { get; set; }

    // Target ART (j)
    public Guid TargetArtId { get; set; }
    public ART? TargetArt { get; set; }

    // Coefficient a_ij (0 to 1)
    public double Coefficient { get; set; }

    public Guid DependencyMatrixId { get; set; }
    public DependencyMatrix? DependencyMatrix { get; set; }
}

public abstract class Node
{
    public Guid Id { get; set; } = Guid.NewGuid();
    public string Name { get; set; } = string.Empty;
    public string? Description { get; set; }
    public int PositionX { get; set; }
    public int PositionY { get; set; }

    // Parent relationship
    public Guid? ParentNodeId { get; set; }
    public Node? ParentNode { get; set; }
    public List<Node> Children { get; set; } = new();

    // Tree reference
    public Guid? DecisionTreeId { get; set; }
    public DecisionTree? DecisionTree { get; set; }

    // Calculated entropy (cached)
    public double? CalculatedEntropy { get; set; }
}

public class DecisionNode : Node
{
    public List<ChanceNode> Alternatives { get; set; } = new();
```

```
}

public class ChanceNode : Node
{
    public List<Probability> Probabilities { get; set; } = new();
}

public class Probability
{
    public Guid Id { get; set; } = Guid.NewGuid();
    public string Alternative { get; set; } = string.Empty;
    public double Value { get; set; } // Must sum to 1.0 for the node

    public Guid ChanceNodeId { get; set; }
    public ChanceNode? ChanceNode { get; set; }
}
```

ДОДАТОК Б
ПРОГРАМА МАГІСТЕРСЬКОЇ КОНФЕРЕНЦІЇ

Результати досліджень, викладені в даній роботі, доповідалися та обговорювалися на 81-ї звітній студентської наукової конференції Одеського національного університету імені І. І. Мечникова [Б.1]. Титульний аркуш та ділянку програми конференції показано на рис. Б.1 та Б.2 відповідно.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ОДЕСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ І. І. МЕЧНИКОВА

П Р О Г Р А М А

81-ї звітної студентської наукової конференції
Одеського національного університету імені І. І. Мечникова
(*присвячується 160-й річниці університету*)

22 – 24 квітня 2025 р.

Одеса
ОНУ імені І. І. Мечникова
2025

Рисунок Б.1 – Титульний аркуш програми конференції

ПІДСЕКЦІЯ "ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ"

Керівник підсекції – докт. техн. наук, проф. Н. Ф. Казакова
Секретар – канд. техн. наук, доц. Б. В. Перелигін

22 квітня 2025 р., 10.00 , дистанційно, вхід за посиланням
https://us02web.zoom.us/j/7863384194?pwd=RUhXLzF0d1NKM0lHaERjci92bk5Edz09
Ідентифікатор: 786 338 4194, Код доступу: 123456

1. Програмні інструменти активізації студентів в умовах онлайн-навчання.
Доповідач – студ. 3 курсу А. Ненахова
Науковий керівник – к.т.н., доц. Т.М. Терещенко

2. Прогнозування академічної успішності студентів на онлайн-курсах.
Доповідач – студ. 3 курсу Ю.Д. Гнатовська
Науковий керівник – к.т.н., доц. Т.М. Терещенко

3. Мобільний Android-застосунок для контролю рівня стресу та занять з медитації.
Доповідач – студ. 5 курсу А.В. Незнайко
Науковий керівник – к.т.н., доц. Т.М. Терещенко

4. Моделі аналізу даних для підсистеми обходу перешкод роботизованою платформою на базі ESP32.
Доповідач – магістр 1 року навчання В. А. Вакарчук
Науковий керівник – д.т.н., доц. С. С. Великодний

5. Інформаційна система для оптимізації комунікації між стейхолдерами на основі аналізу дерев проєктних рішень за методологією Agile.
Доповідач – магістр 1 року навчання А. В. Лапаєв
Науковий керівник – д.т.н., доц. С. С. Великодний

6. Цифрова екосистема для зберігання та розповсюдження українських локалізацій відеоігор.
Доповідач – магістр 1 року навчання С.В. Ковальов
Науковий керівник – к.т.н., доц. Б.В. Перелигін

Рисунок Б.2 – Ділянка програми конференції

Перелік посилань

Б.1. Лапаєв А. В. Інформаційна система для оптимізації комунікації між стейкхолдерами на основі аналізу дерев проєктних рішень // Матеріали 81-ї звітної студентської наукової конференції Одеського національного університету імені І. І. Мечникова. Одеса : ОНУ імені І. І. Мечникова, 2025.