НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«На правах рукопису»
УДК 004.4

«До захисту допущено»
Завідувач кафедри
_____ Євгенія СУЛЕМА
«___»_____2022 р.

Магістерська дисертація

на здобуття ступеня магістра

за освітньо-професійною програмою
«Інженерія програмного забезпечення мультимедійних та інформаційно-пошукових систем»

зі спеціальності 121 Інженерія програмного забезпечення

на тему: «Програмний метод ідентифікації ризиків та затримок у Agile-проєктах»

Виконав:
студент ІІ курсу, групи КП-11мп
Тютюнник Петро Богданович _____

Керівник:
Доцент кафедри ПЗКС, к.т.н.,
Рибачок Наталія Антонівна _____

Консультант з нормоконтролю:
Доцент кафедри ПЗКС, к.т.н., доцент,
Онай Микола Володимирович _____

Рецензент:
С.н.с. МННЦ ІТС, к.т.н., с.н.с.,
Савченко-Синякова Євгенія Анатоліївна _____

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2022 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

Рівень вищої освіти – другий (магістерський)

Спеціальність (спеціалізація) – 121 «Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення мультимедійних та інформаційно-пошукових систем»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Євгенія СУЛЕМА

«___»_____2021 р.

ЗАВДАННЯ
на магістерську дисертацію студенту

Тютюннику Петру Богдановичу

1. Тема дисертації «Програмний метод ідентифікації ризиків та затримок у Agile-проєктах», науковий керівник дисертації Рибачок Наталія Антонівна, доцент кафедри ПЗКС, к.т.н., затверджені наказом по університету від 25.11.2022 р. № 4317-С

2. Термін подання студентом дисертації «14» грудня 2022 р.

3. Об'єкт дослідження: процес ідентифікації ризиків та затримок у Agile-проєктах.

4. Предмет дослідження: методи та алгоритми ідентифікації ризиків та затримок у Agile-проєктах.

5. Перелік завдань, які потрібно розробити:

- дослідити існуючі програмні методи ідентифікації ризиків та затримок у Agile проєктах;
- виділити основні переваги та недоліки існуючих рішень;
- запропонувати гіпотезу щодо покращення результатів існуючих методів;
- сформулювати метод, що підтверджує гіпотезу;
- розробити архітектуру та обрати засоби створення програмного забезпечення;
- провести тестування програмного продукту;
- виділити основні переваги та недоліки програмного продукту;
- провести порівняння розробленого додатку з аналогами;
- здійснити верифікацію результатів дослідження.

6. Перелік графічного (ілюстративного) матеріалу:

- схема роботи запропонованого методу;

- архітектура розробленого ПЗ;
- ERD-діаграма бази даних;
- графік порівняння з базовим методом;
- дерево проблем.

7. Перелік публікацій:
   - Тези доповіді “ Програмний метод ідентифікації ризиків та затримок у Agile-проєктах ”

8. Консультанти розділів дисертації

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Онай М. В., доцент кафедри ПЗКС		

9. Дата видачі завдання «04» жовтня 2021 р.

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1.	Ґрунтовне ознайомлення з предметною галуззю	17.10.2021	
2.	Визначення структури магістерської дисертації; вивчення літератури, пошук додаткової літератури, патентний пошук	05.12.2021	
3.	Робота над першим розділом магістерської дисертації; проведення наукового дослідження	15.02.2022	
4.	Проведення наукового дослідження; робота над другим розділом магістерської дисертації; розроблення програмного забезпечення	05.04.2022	
5.	Проведення наукового дослідження; робота над статтею за результатами наукового дослідження	16.05.2022	
6.	Проведення наукового дослідження; робота над третім розділом магістерської дисертації	15.06.2022	
7.	Завершення роботи над основною частиною магістерської дисертації; підготовка ілюстративного матеріалу; підготовка матеріалів доповіді на конференції ПМК-2022	04.11.2022	
8.	Оформлення текстової і графічної частини магістерської дисертації	05.12.2022	

Студент Петро ТЮТЮННИК

Науковий керівник Наталія РИБАЧОК

РЕФЕРАТ

Актуальність теми. Частка технологічних компаній у останні роки, що використовують хоча б частково методологію Agile досягнуло кількості у 95% в останні роки, при цьому кількість команд розробників програмного забезпечення після переходу на віддалену роботу, що застосовують методологію Agile, зросло більше ніж у 2 рази, до 86% усіх команд. Таке різке збільшення популярності використання методології, створює нові виклики для оцінки ризиків та затримок для команд та клієнтів, що застосовують методологію Agile, яку можна спростити за допомогою створення програмного методу з ідентифікації можливих ризиків та затримок Agile-проєктах. Таким чином, можна зробити висновок, що створення програмного методу для автоматизованої ідентифікації ризиків та затримок у Agile-проєктах, що базується на відповідному методі машинного навчання, є актуальною задачею.

Об'єктом дослідження є процес ідентифікації ризиків та затримок у Agile-проєктах.

Предметом дослідження є методи ідентифікації ризиків та затримок у Agile-проєктах.

Мета роботи: розроблення методу та відповідного ПЗ для ідентифікації ризиків та затримок у Agile-проєктах.

Методи дослідження. В роботі використовуються методи класифікації, методи оптимізації, методи аналізу.

Наукова новизна. Вперше запропоновано модифікацію методу випадкового лісу для врахування даних на основі дат, застосовуючи коефіцієнт відхилення ймовірності ризику, отриманий процедурою прогнозування даних на основі часових рядів Prophet. Отриманий метод підвищив характеристики релевантності влучності, повноти та F-міри на 0,01-0,02 відносно до класичного методу випадкового лісу.

Практична цінність отриманих в роботі результатів полягає в тому, що запропонований метод дає можливість ідентифікувати ризики та

затримки у Agile-проєктах з більшою точністю, оскільки націлений саме на дану сферу та враховує особливості характеру змін даних. Даний метод є методом автоматизованої класифікації, а отже надає можливість створення програмного забезпечення на його основі. В результаті розроблення ПЗ для ідентифікації ризиків та затримок допоможе командам та клієнтам, що використовують методологію Agile, в ідентифікації можливих ризиків та затримок, а також у збільшенні кількості успішних проєктів.

Апробація роботи. Основні положення і результати роботи були представлені та обговорювались на науковій конференції магістрантів та аспірантів «Прикладна математика та комп'ютинг» ПМК-2022.

За результатами роботи написана стаття «SOFTWARE METHOD OF IDENTIFYING RISKS AND DELAYS IN AGILE PROJECTS», яка прийнята до опублікування у журнал категорії Б «Control Systems and Computers».

Структура та обсяг роботи. Магістерська дисертація складається з вступу, п'яти розділів, висновків та додатків.

У вступі надано загальну характеристику роботи, описано актуальність задачі та проблеми, які вона може вирішити. Також висунуто мету, об'єкт та предмет дослідження.

У першому розділі описано задачу ідентифікації ризиків та затримок у Agile-проєктах та розглянуто й описано базові методи класифікації.

У другому розділі описано та проаналізовано зміст завдання у Agile-проєктах та описано параметри, які є можливими факторами ризику. Оцінено чотири базових метода машинного навчання та запропоновано й описано модифікацію для методу випадкового лісу використовуючи процедуру Prophet.

У третьому розділі обґрунтовано засоби реалізації програмного забезпечення на основі запропонованого методу. Наведено архітектуру розроблюваного програмного забезпечення та описано складові модулі системи.

У четвертому розділі проведено оцінку існуючих та запропонованого методів. Запропонований метод показав на 0,01-0,02 характеристики релевантності вищі ніж базовий метод випадкового лісу, що має найкращі результати серед досліджуваних немодифікованих методів.

У п’ятому розділі проведено опис бізнес-моделі, огляд існуючих аналогів системи та проведено аналіз ринкових можливостей запуску стартап-проєкту.

У висновках проаналізовано отримані результати роботи.

Робота виконана на 80 аркушах, містить 8 додатків та посилання на список використаних літературних джерел з 45 найменувань. У роботі наведено 14 рисунків та 7 таблиць.

Ключові слова: метод класифікації, метод випадкового лісу, Agile, ризики та затримки у Agile-проєктах.

ABSTRACT

Theme urgency. The proportion of technological companies in recent years, which has used at least partially Agile methodology has reached 95% in recent years, with the number of software developers after the transition to remote work used by Agile methodology has increased more than 2 times, to 86 % of all teams. Such a sharp increase in the popularity of the use of methodology, creates new challenges to assess risks and delays for commands and clients using Agile methodology, which can be simplified by creating a software method for identifying possible risks and delays. Thus, it can be concluded that the creation of a software method for automated risk identification and delays in Agile projects based on the appropriate method of training machine is a topical task.

Object of research is a process of identifying risks and delays in Agile projects.

Subject of research are methods and algorithms of identifying risks and delays in Agile projects.

Research objective: development of a method and corresponding software for identifying risks and delays in Agile projects.

Research methods. Classification methods, optimization methods, analysis methods.

Scientific novelty: A modification of the random forest method to account for date-based data is proposed for the first time, using a risk probability deviation coefficient derived from Prophet's time-series data forecasting procedure. The resulting method increased the relevance characteristics of accuracy, completeness, and F-measure by 0.01-0.02 relative to the classic "Random Forest" method.

Practical value of the obtained results is that the proposed method makes it possible to identify the risks and delays in the Agile projections with greater accuracy, since it is aimed at this area and considers the peculiarities of the nature of data changes. This method is a method of automated classification, and therefore allows the creation of software based on it. As a result of the

development of software to identify risks and delays will help teams and clients using Agile methodology in identifying possible risks and delays, as well as increasing the number of successful projects.

Approbation. The basic points and outcomes of the research have been presented and discussed at the scientific conference for students and postgraduates «Applied mathematics and computing» PMK-2022».

Based on the results of the work, the article «SOFTWARE METHOD OF IDENTIFYING RISKS AND DELAYS IN AGILE PROJECTS» was written, which was accepted for publication in the category B magazine «Control Systems and Computers».

Structure and content of the thesis. The master thesis consists of the introduction, five chapters, conclusions and appendixes.

The introduction provides a general description of the work, describes the relevance of the task and the problems it can solve. The goal, object and subject of the research are also put forward.

In the first chapter the task of identifying risks and delays in Agile projects is described and basic classification methods are described.

In the second chapter the content of tasks in Agile projects is described and analyzed. Parameters, that are possible risk factors, are described. Four basic machine learning methods are evaluated and a modification to the random forest method using the Prophet procedure is proposed and described

In the third chapter the means of software implementation based on the proposed method are described. The architecture of the developed software is given, and the component modules of the system are described.

In the fourth chapter an assessment of the existing and proposed methods is carried out. The proposed method showed 0.01-0.02 higher relevance characteristics than the basic random forest method, which has the best results among the studied unmodified methods.

In the fifth chapter description of the business model, an overview of existing analogues of the system were provided, and an analysis of the market opportunities for launching a startup project was carried out.

In the conclusions the general conclusions on the presented thesis are given; the obtained results are analyzed.

The thesis is presented in 80 pages, it contains 8 appendixes and 45 references to the used information sources. 10 figures and 7 tables are given in the thesis.

Key words: classification, random forest, Agile methodology, risks and delays in Agile Projects.

ЗМІСТ

СПИСОК ВИЗНАЧЕНЬ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

ПЗ – програмне забезпечення.

Ризик – невизначена подія або умова, які, якщо вони відбуваються, мають позитивний або негативний вплив на цілі проєкту. Серед негативних впливів найбільш поширеними є затримка в часі виконання проєкту або необхідність в додаткових ресурсах.

HR – human resources, сукупність робітників з кваліфікацією до розробки або підтримки продукту або сервісу.

Agile – клас методологій розробки програмного забезпечення, що базується на ітеративній розробці, в якій вимоги та розв'язки еволюціонують через співпрацю між багатофункціональними командами здатними до самоорганізації.

KANBAN – метод управління розробкою програмного забезпечення з наголосом з доставкою якраз вчасно, та униканні перевантаження членів команди.

Scrum – підхід управління проєктами для гнучкої розробки програмного забезпечення.

Задача класифікації – це формалізована задача, яка містить множину об'єктів, поділених певним чином на класи. Задана скінченна множина об'єктів, для яких відомо, до яких класів вони належать. Ця множина називається вибіркою. До якого класу належать інші об'єкти невідомо. Необхідно побудувати такий алгоритм, який буде здатний класифікувати довільний об'єкт з вихідної множини.

Машинне навчання – це підгалузь штучного інтелекту в галузі інформатики, яка часто застосовує статистичні прийоми для надання комп'ютерам здатності «навчатися» (тобто, поступово покращувати продуктивність у певній задачі) з даних, без того, щоби бути програмованими явно.

Atlassian JIRA – система відстеження помилок, призначена для організації спілкування з користувачами, і для управління проєктами.

Розроблена компанією Atlassian в 2002 році.

JVM – Java Virtual Machine, набір комп'ютерних програм та структур даних, що використовують модель віртуальної машини для виконання інших комп'ютерних програм чи скриптів.

API – Application Programming Interface, інтерфейс, який задає взаємодію між декількома програмними застосунками або між посередниками програмного забезпечення та пристрою.

Фреймворк – програминий засіб, що призначений для спрощення роботи з певним програмним забезпеченням, шляхом створення рівня абстракції над внутрішніми низькорівневими викликами.

DOM – Document Object Model, Об'єктна модель документа – специфікація прикладного програмного інтерфейсу для роботи зі структурованими документами (як правило, документами XML).

TypeScript – мова програмування, представлена Microsoft восени 2012; позиціонується як засіб розробки вебдодатків, що розширює можливості JavaScript.

БД – база даних.

СУБД – система управління базами даних.

SQL – декларативна мова запитів, що застосовується для створення, модифікації та управління даними в реляційній БД.

NoSQL – Not Only SQL, окремий тип СУБД, що підтримує не лише SQL мову для управління.

BSON – Binary JavaScript Object Notation, формат електронного обмінну цифровими даними.

JSON – JavaScript Object Notation, формат представлення інформації для обміну між програмними модулями.

HTTP – Hyper Text Transfer Protocol, протокол передачі гіпер-текстових документів – протокол передачі даних, що використовується в комп'ютерних мережах.

UUID – стандарт ідентифікації, який використовується при створенні програмного забезпечення.

MongoDB Atlas – хмарний сервіс, що надає послуги автоматизованої нереляційної бази даних на основі ядра СУБД MongoDB.

ВСТУП

За останні роки неймовірно високим темпом піднімаються прийняття та застосування методології Agile. Станом на 2021 рік, відповідно до опитування “State of Agile”, цією методологією користуються хоча б частково до 95% технологічних компаній [1]. При цьому за рік пандемії частка лише команд розробників ПЗ, що працюються за методологією Agile, збільшилася з 37 відсотків до 86, що більше ніж удвічі. Такі ж показники відносного росту спостерігаються також і для команд, які зазвичай не застосовували методи Agile, як то маркетинг, HR та фінанси. Хоча є багато переваг прийняття цієї методології, також є і вагомі складнощі, що спостерігаються при переході на дану методологію та проявляються при роботі з нею.

Основною причиною невдалих Agile-проєктів називають ризики, які не були виявлені на етапі планування, і результуючі великі затримки [2]. При цьому найбільш популярні існуючі методи виявлення ризиків і затримок не є автоматизованими і, отже, більш схильні до людських помилок. Часто проєктні команди не можуть вчасно визначити необхідні ризики, що призводить до провалу проєктів [2].

Таким чином є необхідність у розробці програмного забезпечення для ідентифікації ризиків і затримок у Agile-проєктах, що буде допомагати командам таких проєктів у визначенні можливих ризиків і заьримок серед завдань у методології Agile. Відповідно для реалізації такого ПЗ потрібно розроблення теоретичного підґрунтя у вигляді програмного методу ідентифікації ризиків та затримок у Agile-проєктах.

Отже, метою даної наукової роботи є підвищення характеристик влучності, повноти та F-міри для розроблюваного методу ідентифікації ризиків та затримок у Agile-проєктах у порівнянні з існуючими методами вирішення даної проблеми шляхом розроблення та програмної реалізації відповідного методу з урахуванням специфіки Agile-проєктів.

Об’єкт даної роботи – процес ідентифікації ризиків та затримок у

Agile-проєктах.

Предметом даної роботи є методи та алгоритми ідентифікації ризиків та затримок у Agile-проєктах.

1. ОГЛЯД ІСНУЮЧИХ МЕТОДІВ ІДЕНТИФІКАЦІЇ

1.1. Опис задачі ідентифікації ризиків та затримок

Методологія Agile має значну кількість переваг та принципів, що дозволили їй досягнути високого рівня використання у 86% серед команд, що розробляють ПЗ [1]. Одна з цих переваг це гнучкість до зміни вимог, що досягається за рахунок постійному ітераційному або інкрементованому створенню цінного ПЗ, із додавання нового функціоналу та покращень існуючого продукту у циклах по кілька тижнів.

Для досягнення даної переваги найбільш популярні парадигми, як SCRUM або KANBAN, використовується розбиття вимог на малі завдання, що є окремими одиницями роботи, які можуть виконуватися одним членом команди, та додаватись чи видалятись протягом життєвого циклу проєкту. У цілому виконання Agile-проєкту можна розглядати як виконання множини малих завдань, кожне з яких може бути затриманим та бути ризиком. Велика кількість затримок серед завдань може стати причиною для невдачі Agile-проєкту. Зважаючи на це, задачу ідентифікації ризиків і затримок у Agile-проєктах розглядатимемо як визначення чи є кожне окреме завдання в проєкті можливим ризиком та затримкою.

Кожне завдання є окремим об’єктом із різними характеристиками та може бути поділено на два класи:

- завдання є можливим ризиком;
- завдання не є можливим ризиком.

Задачею ідентифікації ризиків і затримок у Agile-проєктах є знаходження до якого з двох класів належить об’єкт завдання, тобто задача ідентифікації є формалізованою задачею бінарної класифікації [3].

Бінарна класифікація – це формалізована задача, в якій довільний об’єкт даних класифікується як один з двох відомих класів [3]. У цій задачі, задана вибірка – скінченна множина об'єктів, для яких відомо, до якого з класів вони належать. Необхідно побудувати такий метод, який буде здатний класифікувати довільний об'єкт до одного з двох класів.

Прикладами типових задач із класифікації є:

- медичне тестування, що визначає наявність або відсутність певного захворювання;
- контроль якості на виробництві, що визначає відповідність або невідповідність виробу вимогам специфікації;
- інформаційний пошук, що визначає включати чи не включати інформаційний ресурс у результат пошуку.

Зазвичай у задачі бінарної класифікації два класи не є симетричними за обсягом відмінних наборів даних з кожного класу та за наслідками помилкової класифікації [3]. Наприклад, у даній роботі кількість завдань, що були затримані, складають лишень 16,7% серед усіх завдань, що були зібрані із проєктів з відкритим кодом. При цьому класифікація завдання, що не є ризиком, ризиком є менш критичною помилкою для Agile-проєкту, ніж зворотна помилка.

У контексті цієї роботи метод повинен працювати над визначення чи є наявне у Agile-проєкті завдання можливим ризиком чи ні, що є двома класами у задачі бінарної класифікації.

1.2. Огляд методів класифікації

Існують різні підходи до виконання класифікації і відповідно велика кількість способів. Найбільш вживаними є способи машинного навчання, а саме спосіб навчання з учителем.

Навчання з учителем або кероване навчання – це спосіб машинного навчання, у ході якого система примусово навчається за допомогою наявної множини прикладів «стимул-реакція» з метою визначення «реакції» для «стимулів», які не належать до наявної множини прикладів [4].

Між входами та еталонними виходами в навчанні з учителем може існувати залежність, яка не відома. Відома лише кінцева сукупність прецедентів – пар «стимул-реакція», що зветься навчальною вибіркою. На основі цих даних потрібно відновити залежність, тобто побудувати

алгоритм, здатний для будь-якого об'єкта видати досить точну відповідь. Для вимірювання точності відповідей, застосовуються різні характеристики релевантності, як влучність та повнота.

Формально задача навчання з учителем може бути сформульована як визначення для сукупності стимулів *x* та реакцій *y* такої залежності $f(x)$, що отриманий результат не покидає межі заданої похибки *e*:

$$f(x) - e \leq y \leq f(x) + e .$$

Навчання з учителем може використовувати велику кількість різних типів вхідних даних для визначення залежності до реакцій:

1. Ознаковий опис. Кожен об'єкт описується набором своїх характеристик, що називаються ознаками. Ознаки можуть бути числовими або нечисловими, наприклад булеві значення. Найпоширеніший тип вхідних даних для навчання із вчителем та способів машинного навчання в цілому. У рамках цієї роботи застосовуватиметься саме цей тип вхідних даних.
2. Матриця відстаней між об'єктами. Кожен об'єкт описується відстанями до всіх інших об'єктів навчальної вибірки. З цим типом вхідних даних працюють такі методи, як метод k-найближчих сусідів, метод парзенівського вікна, метод потенційних функцій.
3. Зображення або відеоряд.
4. Часовий ряд або сигнал. Цей тип даних є послідовністю вимірювань в часі. Кожен вимір може представлятися числом, вектором, а в загальному випадку – ознаковим описом досліджуваного об'єкта в даний момент часу. Для цих типів даних застосовують особливі методи машинного навчання, як процедура Prophet, які відмінні від тих методів, що використовуються для інших типів ознак.

Також можуть зустрічатися складніші типи даних для навчання моделей машинного навчання. Наприклад вхідні дані можуть подаватися у вигляді графів, текстів, результатів запитів до бази даних у форматі SQL

тощо. Як правило, вони приводяться до першого або другого випадку шляхом попередньої обробки даних та виділяння ознак. Наприклад у цій роботі взаємозв’язки між проблемами у системі відстеження помилок Atlassian JIRA, які є графами, були зведенні до ознак, як кількість пов’язаних проблем або кількість проблем, які блокуються цією проблемою. Більш детально ознаки, що були виділені для проблем системі відстеження помилок Atlassian JIRA та були використані в цій роботі для навчання моделей машинного навчання описанні в підрозділі 2.1.

Серед реакцій у методах навчання можна виділити наступні типи відповідей:

- множина можливих відповідей нескінченна;
- множина можливих відповідей скінчена;
- відповідь вказує на майбутню поведінку процесу або явища.

Коли множина можливих відповідей нескінченна, способом навчання з учителем можуть вирішуватись задачі апроксимації та регресії.

Коли множина можливих відповідей скінченна, способом навчання з учителем можуть вирішуватись задачі розпізнавання образів та класифікації.

Коли відповідь вказує на майбутню поведінку процесу або явища, способом навчання з учителем може вирішуватись задача прогнозування.

На протиставлення до способу навчання з учителем, існує спосіб навчання без вчителя. У даному способі, при вирішенні задач машинного навчання, випробовувана система спонтанно навчається виконувати поставлене завдання, без втручання з боку експериментатора. Спосіб навчання без вчителя дозволяє для задач, де є відома навчальна вибірка, виявляти внутрішні взаємозв'язки, залежності, закономірності, що існують між об'єктами вибірки [4].

Оскільки задача з ідентифікації ризиків та затримок в Agile-проєктах є задачею класифікації, для якої відома множина ознак у вигляді навчальної вибірки та множина відповідей є скінченною, найбільш доцільно

застосовувати саме спосіб навчання з вчителем для вирішення описаної задачі.

Виділимо основні методи машинного навчання, які навчаються способом навчання з вчителем та можуть використовуватись для вирішення задачі автоматизованої бінарної класифікації:

- дерево рішень;
- метод випадкового лісу;
- штучна нейронна мережа;
- наївний баєсівський класифікатор.

Розглянемо детальніше основні автоматизовані методи бінарної класифікації.

Дерево рішень

Дерево рішень – це непараметричний алгоритм керованого навчання. Використовує деревоподібну, ієрархічну структуру [5]. Складається з вузлів, гілок, внутрішніх вузлів, вузлів листових. Має вигляд блок-схеми. По своїй суті є аналогом способу прийняття рішень людиною: «якщо авто червоне – відповідно воно красиве».

Алгоритм навчання вибірково розподіляє усю множину на підмножини можливих рішень і з кожним рівнем відкидає нерелевантні можливості. Без відсікання отримані дерева зазвичай є занадто деталізованими, що призводить до перенавчання, та подальше застосування яких дає значну кількість помилок. Після відсікання нерелевантних гілок, залишається лише найпідпорядкованіший варіант дерева рішень для кожного з вузлів вище.

Серед переваг методу виділяються:

- підтримка методом одночасно задачі класифікації та регресії;
- висока швидкодія при використанні великої кількості даних;
- потребує малу вибірку для тренувань моделі;
- отримана модель легка для розуміння;

- легко знайти помилки в отриманій моделі.

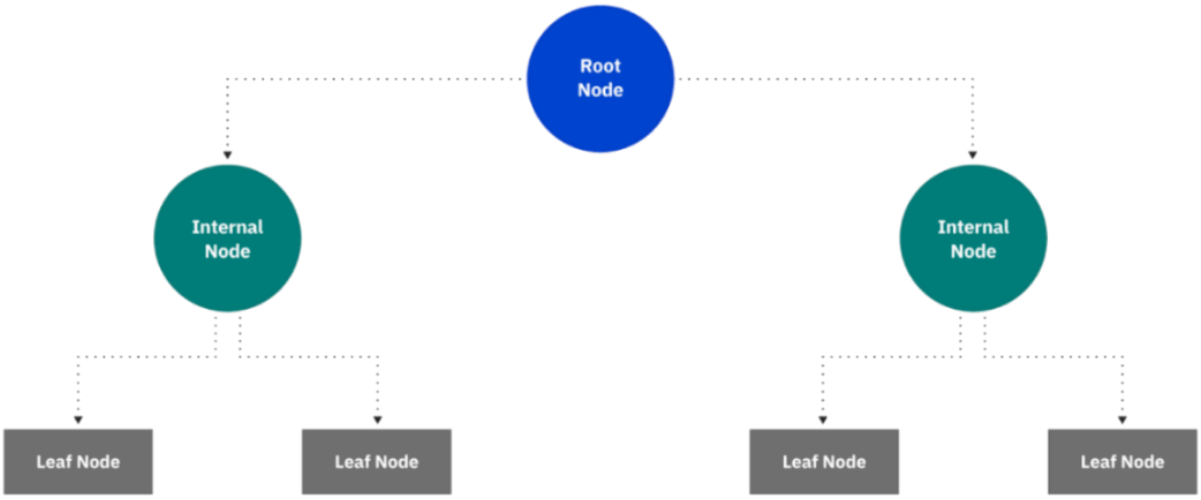


Рис. 1.1. Візуалізація методу дерево рішень

Недоліками методу є:

- часті перенавчання усієї моделі для об’єктів з великою кількістю параметрів;
- структура моделі може сильно змінюватися під час навчання;
- має лише одне модливе рішення, що обмежує вибір найбільш оптимального рішення;
- категоризація відбувається на основі вагів, відповідно неможливість специфікації за певними атрибутами.

Метод випадкового лісу

Метод випадкового лісу – є розширенням методу дерева рішень, в якому об'єднуються вихідні дані декількох дерев мережі замість одного дерева у класичному і видається єдиний результат [6]. Використовується для вирішення проблем класифікації та регресії. Кожне з дерев що береться за основу виконує базову функцію дерева рішення, листком позначається кожне з остаточних рішень. Після класифікації усі рішення з позитивним

результатом будуть відправленні до однойменної гілки. Об'єднання дерев підвищує точність класифікації.

Для методу випадкового лісу створюються n підвибірок початкової навчальної вибірки, в якій частина прикладів повторюється, а приблизно третина не входять у підвибірку, де n – кількість піддерев, що використовуються у відповідній реалізації методу. Для кожної з підвибірок обирається випадкові m ознак серед M усіх ознак набору. Дана модифікація дозволяє методу випадкового лісу обробляти великі набори даних, що мають значну кількість ознак. Негативним ефектом цієї модифікації є великий розмір отриманої моделі. У середньому модель методу випадкового лісу в n разів більша за модель методу дерева рішень де n – кількість піддерев.

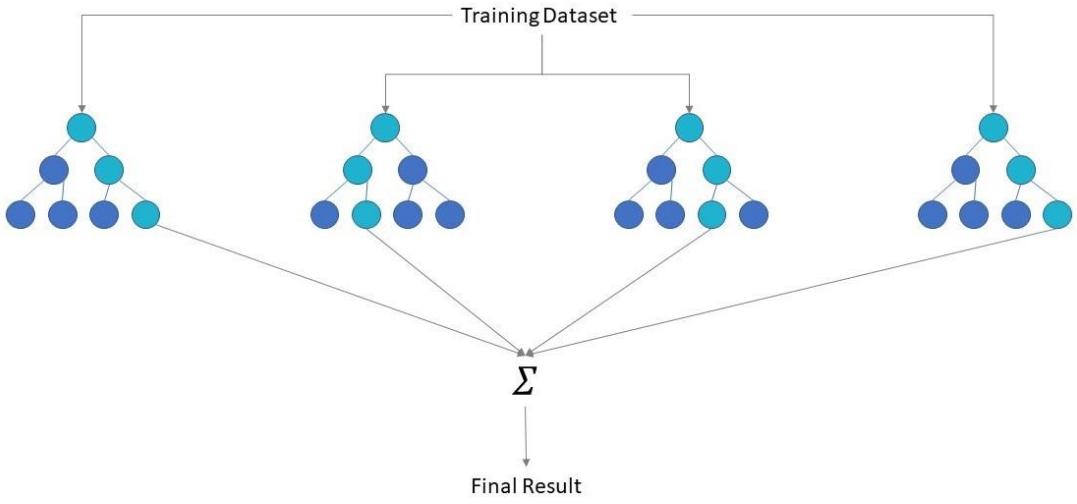


Рис. 1.2. Візуалізація методу випадкового лісу

Перевагами методу є:

- дані з великим числом ознак і класів обробляються ефективніше ніж іншими методами;
- нечутливість до масштабування та монотонних перетворень значень ознак у цілому;
- є методи оцінювання значущості окремих ознак в моделі;
- здатність працювати паралельно в багато потоків, що збільшує

швидкодію для багатопотокових систем;

- однаково добре обробляються як безперервні, так і дискретні ознаки.

Недоліками методу є:

- алгоритм схильний до перенавчання на завданнях з великою кількістю шумів;
- великий розмір отримуваних моделей у порівнянні з моделями інших методів.

Штучна нейронна мережа

Штучні нейронні мережі – це обчислювальні системи, за основу яких узято біологічні нейронні мережі, що складають мозок тварин [7]. Такі системи навчаються задач та поступово покращують свою продуктивність на них, розглядаючи приклади, загалом без спеціального програмування під задачу.

Глибоке навчання – це спосіб підвищення ефективності мережі за допомогою збільшення кількості прихованих шарів.

Кожна мережа використовує трьохшарову систему розподілення нейронів. Навчання виконується за допомогою розподілення вагів і великої кількості тестових ітерацій. Задана каскадна модель обирає за таким принципом дані до моменту досягнення прийнятних результатів на рівні вихідного шару. Тип передачі даних зменшує вплив будь-якої змінної на вихід вузла і останнього шару відповідно. Кожен нейрон аналізує вхідні аргументи з урахуванням ваги, підставляючи їх суму у функцію активації.

Перевагами методу є:

- точність отриманих результатів підвищується з часом;
- потужний інструмент для кластеризіції даних;
- швидке виконання задач готовою моделлю.

Недоліками методу є:

- швидкість навчання;

- велика вибірка даних для тренувань;
- велика вірогідність отримати проблему локального мінімуму;
- важкість у пошуку помилкового нейрону у разі помилки у тренуванні.

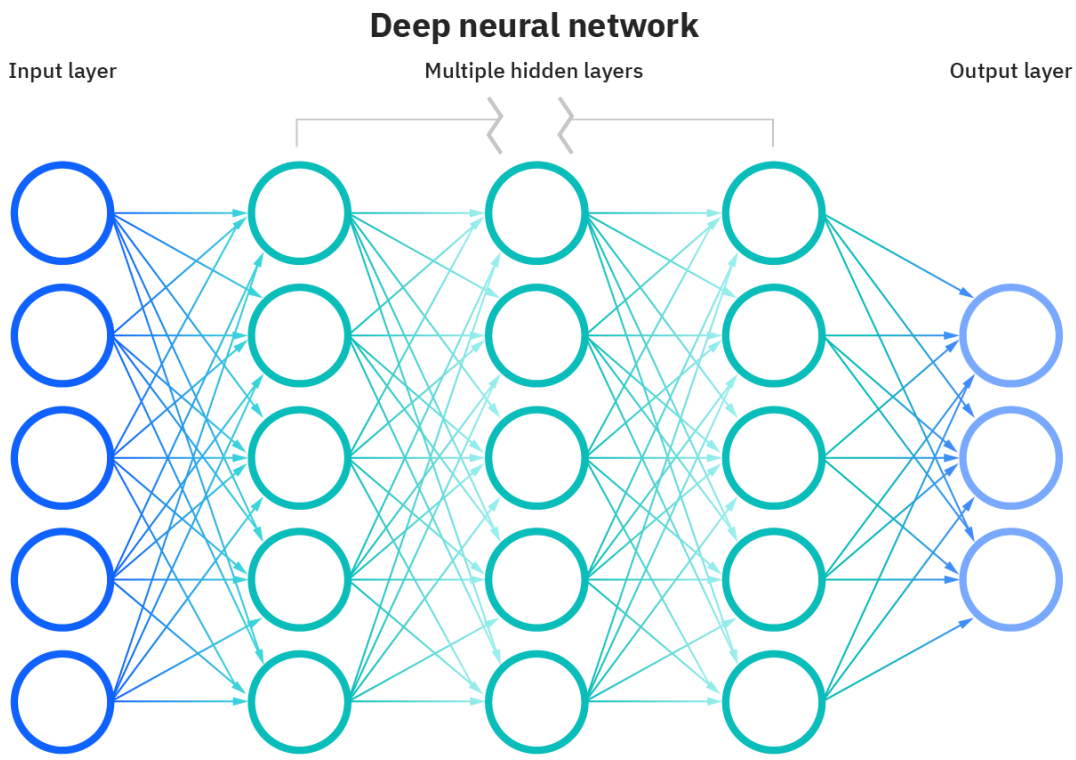


Рис. 1.3. Візуалізація штучної нейронної мережі

Наївний баєсівський класифікатор

Наївний баєсівський класифікатор – це алгоритм керованого навчання базований на теоремі Баєса [8]. Ґрунтується на побудові ймовірнісних прогнозів. Кожна ознака слугує в даній методі ідентифікацією вихідного результату, вони є незалежними одна від одної. Принцип роботи: перетворення даних у табличний вигляд, відповідне формулювання вірогідностей, обчислення апостеріорі імовірності за допомогою теореми Баєса. Видає завжди найоптимальніший результат, в протилежному випадку неоднозначність буде охарактеризована кількісно. Результат не може бути поліпшений.

Перевагами методу є:

- швидке розв'язання задач класифікації;
- може використовуватись для оцінки текстових характеристик;
- помірні вимоги отриманої моделі до пам'яті.

Недоліками методу є:

- не враховує взаємозв'язок між декількома ознаками;
- складність задачі має прямий вплив на кількість змінних (атрибутів);
- експоненціальний ріст складності обчислень до кількості характеристик досліджуваних об’єктів;
- зниження точності у випадку залежності між атрибутами;
- вибірка має містити усі комбінації змінних.

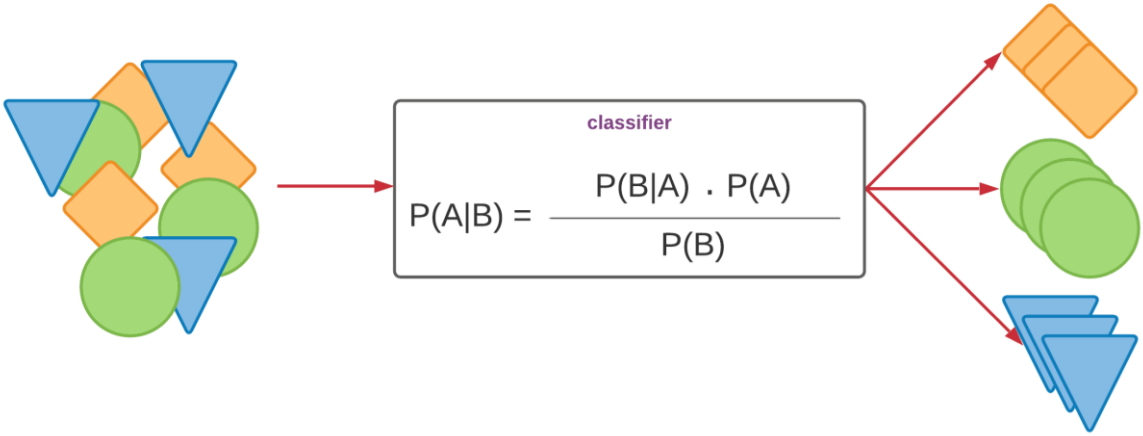


Рис. 1.4. Візуалізація методу наївного баєсівського класифікатору

1.3. Висновки до розділу

У рамках цього розділу було описано задачу ідентифікації ризиків і затримок у Agile-проєктах, яка є задачею бінарної класифікації та вирішується методами машинного навчання.

Було розглянуто та описано спосіб машинного навчання навчання з вчителем, що використовується для вирішення задачі класифікації. Було розглянуто та описано чотири базові методи машинного навчання серед

яких: метод випадкового лісу, наївний баєсівський класифікатор, дерево рішень та штучні нейроні мережі, та які будуть оцінені у цій роботі.

2. МЕТОД ІДЕНТИФІКАЦІЇ РИЗИКІВ І ЗАТРИМОК

2.1. Аналіз змісту завдань

Команди, які використовують методологію Agile, розбивають свою задачу з проєкту на завдання, що містять інформацію про незначну частину роботи, яку повинні виконати члени команди, та інформацію про прогрес у виконанні цієї роботи. Відповідна інформація може бути використана для ідентифікації, чи є відповідне завдання можливим ризиком та затримкою в Agile-проєкті, та для навчання та тестування методів машинного навчання. Проаналізуємо зміст завдань більш детально на прикладі проблем з системи відстеження помилок Atlassian JIRA, данні з якої використовувалися при навчанні моделей машинного навчання у даній роботі.

Завдання у системі JIRA складаються з полів, які є типовими для усіх користувачів даної системи, та полів, що додаються спеціальними плагінами. Наприклад, команда Agile-проєкту може використовувати різні системи контролю версій, такі як GitHub або Bitbucket, де кожна з систем має свій власний плагін, поля для якого є унікальними. Зважаючи на те, що у даній роботі використовуються дані з 4 різних проєктів із відкритим кодом, що описанні в підрозділі 4.1, для вибору параметрів для навчання моделей використано лише поля, що є типовими для Atlassian Jira.

Основними характерниками проблеми є назва, опис, тип, пріоритет завдання, репортер, який створив завдання, розробник який виконує завдання, коментарі під завданням, історія змін завдання. Більшість із цих характеристик є текстовими та не можуть бути використані при навчанні моделей машинного навчання. Тому було визначено наступні параметри, які можуть бути факторами ризику завдання:

1. Час обговорення – це період, який команда витрачає на пошук рішення проблеми. Agile-проєкт можна розглядати як мережу заходів, де кожна діяльність реєструється як проблема, час на виконання якої впливає на загальний графік проєкту. Для проблеми, на вирішення якої потрібен значний час, це може

призвести до затримок.

2. Час очікування – час очікування вказує на час, коли проблема очікує на вирішення, наприклад очікування, поки призначений розробник виконає дії. Ненормальний час очікування є ознакою того, що проблема відкладається через відсутність співпраці команди, або що ніхто не хоче мати справу з проблемою [9]. Час очікування проблеми починається з моменту призначення відповідної особи до виконання дій для вирішення проблеми.
3. Тип – кожній проблемі в системі JIRA призначається тип (наприклад, завдання, помилка, нова функція, вдосконалення чи документація), який вказує на характер завдання, пов’язаного з вирішенням проблеми (наприклад, виправлення помилки або впровадження нової функції).
4. Кількість разів повторного відкриття проблеми – попередні дослідження з ідентифікації ризиків [10] вказують, що повторні відкриття завдань (тобто повтори в життєвому циклі проблеми) вважаються фактором погіршення загальної якості ПЗ. Це призводить до додаткових і непотрібних переробок, що сприяє затримкам. Проблема відкривається повторно з кількох причин, наприклад, якщо вона фактично не була належним чином вирішена й потребує доопрацювання.
5. Пріоритет – порядок, в якому питання має розглядатися відносно інших питань. Наприклад, проблеми з пріоритетом блокера (проблема, яка блокує інші проблеми) повинна розглядатися раніше за інші завдання.
6. Зміна пріоритету – кількість разів пріоритет проблеми було змінено. Зміна пріоритету проблеми може вказувати на зміну її складності. Наприклад, у деяких дослідженнях [11] використовують дану зміну пріоритету як функцію для прогнозування помилки блокування.

7. Кількість коментарів – кількість коментарів від розробників під час обговорення може вказувати на ступінь співпраці команди [12]. У минулих дослідженнях з теми ідентифікації ризиків і затримок [13], було знайдено, що кількість коментарів впливає на час вирішення помилок: помилки з двома-шістьма коментарями зазвичай вирішуються швидше, ніж помилки з менш ніж двома коментарями та помилки з більш ніж шістьма коментарями.
8. Кількість виправлених версій – вказує версії, для яких проблему було або буде виправлено. Проблеми з великою кількістю версій виправлень потребують більшої уваги з точки зору розробки, тестування та інтеграції. Також потрібен інтенсивний процес перевірки, щоб переконатися, що вирішення проблеми не викликає нових проблем для кожної версії виправлення.
9. Кількість версій з проблемою – вказує на кількість версій, в яких проблему було знайдено. Кількість уражених версій є індикатором потенційного ризику, наприклад потрібно докласти більше зусиль, щоб вирішити проблему з великою кількістю уражених версій.
10. Кількість пов’язаних проблем – вказує на кількість пов’язаних проблем. Зв’язування проблем дозволяє командам створювати асоціації між проблемами. Наприклад, проблема може дублювати іншу, або її вирішення може залежати від інших проблем. Існує кілька типів посилань на проблеми: стосується, дублює та блокує.
11. Кількість проблем, які блокуються цією проблемою – блокування є одним із типів відношення між проблемами. Фактором ризику є кількість проблем, які блокуються цією проблемою. Цей тип залежності проблеми вказує на складність вирішення проблеми, оскільки вона безпосередньо впливає на прогрес інших проблем.
12. Кількість проблем, які блокують цю проблему – це кількість інших проблем, які блокують вирішення цієї проблеми. Вирішення великої кількості проблем із блокерами складніше, оскільки всі

проблеми з блокером потрібно виправляти заздалегідь. Таким чином, кількість проблем з блокерами впливає на час, який необхідний на вирішення проблеми [11].

13. Кількість зміни опису – скільки разів опис проблеми змінювався. Опис проблеми важливий для всіх зацікавлених сторін проблеми. Зміна опису проблеми вказує на те, що проблема нестабільна, а також може спричинити плутанину та непорозуміння, а отже, це можливий фактор ризику.
14. Репутація репортера – відносна оцінка репутації репортера, члена команди, що створив завдання. Фактор репутації репортера досліджувалася в існуючих роботах з ідентифікації можливих ризиків та затримок. Наприклад, було виявлено, що помилки, які повідомляють члени команди з вищою репутацією привертають увагу швидше за інші проблеми [14] та з меншою ймовірністю будуть повторно відкриті [10]. У контексті ідентифікації відкладених проблем репутація репортера може бути одним із факторів ризику, оскільки репортери з низькою репутацією можуть писати погані звіти про проблему, що може призвести до більш тривалого часу для вирішення проблеми [15]. У цій роботі використовується визначення репутації репортера, запропоноване в роботі Hooimeijer та Weimer [14]:

$$reputation(D) = \frac{|opened(D) \cap fixed(D)|}{|opened(D)| + 1} .$$

Репутація репортера *D* вимірюється як відношення кількості проблем, які репортер *D* відкрив і вони були виправлені, до кількості проблем, які репортер *D* відкрив плюс один.

15. Робоче навантаження розробника – кількістю відкритих проблем, призначених розробнику за один раз. Робоче навантаження розробника є відображенням якості планування ресурсів, що має вирішальне значення для успіху проєкту. Відсутність планування

ресурсів має наслідки для невдач проєкту [16], а робоче навантаження розробника може мати значний вплив на хід проєкту [17]. Робоче навантаження розробника (повторно) обчислюється одразу після того, як розробнику було призначено проблему.

16. Відсоток проблем із затримкою, з якими займався розробник – відсоток відкладених проблем серед усіх проблем, призначених розробнику. Члени команди не мають спеціальних навичок, необхідних для проєкту, а недосвідчені члени команди є однією з головних загроз перевиконання графіка [18]. Команди, які складаються з некомпетентних розробників, можуть бути причиною затримки проєкту [9]. З іншого боку, останні дослідження показали, що найкращі розробники часто створюють найбільшу кількість помилок, оскільки вони часто вибирають або отримують найскладніші завдання [19]. Це явище також може стосуватися відкладених проблем: найкращі розробники можуть отримати найбільше/найскладніші проблеми, і тому їм потрібно найбільше часу, щоб їх вирішити. У розробника може виникнути велика кількість відкладених проблем, оскільки він або вона є розробником-експертом, якому завжди доручають вирішення складних питань.

Для визначених параметрів було застосовано l_1 логістичну регресію, використовуючи функцію логарифмічної правдоподібності для оцінки впливу кожного з запропонованих параметрів проблем на ймовірність, що проблема є ризиком.

В l_1 логістичній регресії при негативній кореляції параметру на результат отримуються від’ємні цифри, у той час при позитивній – додатні. При відсутності впливу значення функцію правдоподібності прямує до нуля.

Для оцінки впливу кожного з запропонованих параметрів проблем

було використано набір даних створений з проблем проєктів з відкритим кодом, що описаний у підрозділі 4.1.

Результати логістичної регресії представлені в таблиці 2.1.

Таблиця 2.1

Результати логістичної регресії для параметрів проблем з проєктів з відкритим програмним кодом

Параметр	Результат логістичної регресії
Час обговорення	–6,974
Час очікування	–4,689
Тип: Баг	–0,904
Тип: Документація	1,16
Тип: Покращення	0,247
Тип: Новий функціонал	0,83
Тип: Історія	–0,528
Тип: Підзавдання	1,637
Тип: Завдання	0,783
Пріоритет: Блокер	–0,748
Пріоритет: Критичний	–0,947
Пріоритет: Вагомий	–0,901
Пріоритет: мінімальний	–0,704
Пріоритет: другорядний	–0,764
Кількість разів зміни пріоритету	1,434
Кількість разів повторного відкриття проблеми	3,016
Кількість коментарів	2,963
Кількість виправлених версій	2,207

Продовження табл. 2.1

Кількість версій з проблемою	–3,201
Кількість пов’язаних проблем	1,415
Кількість проблем, які блокуються цією проблемою	0,216
Кількість проблем, які блокують цю проблему	1,935
Кількість зміни опису	1,678
Репутація репортера	–0,683
Робоче навантаження розробника	1,791
Відсоток проблем із затримкою, з якими займався розробник	3,497

2.2. Оцінка ефективності існуючих методів

Для оцінки ефективності існуючих методів машинного навчання для задачі ідентифікації ризиків і затримок у Agile-проєктах будемо використовувати наступні характеристики релевантності [20]:

- Влучність – частка правильно прогнозованих результатів, як позитивних, так і негативних;
- Повнота – вимірює частку істинно позитивних, що є визначеними правильно;
- F-міра – одна з мір точності тесту. Обчислюється використовуючи влучність та повноту тесту, як середнє гармонійне двох названих характеристик релевантності. Найвищим можливим значенням *F*-міри є 1, що настає при ідеальних влучності та повноті, тобто зі значеннями 1, а найнижчим можливим значенням є 0, коли влучність або повнота є нульовими.

Ці характеристики обчислюємо за наступними формулами:

$$P = \frac{t_p}{t_p + f_p},$$

$$R = \frac{t_p}{t_p + f_n},$$

$$F = 2 \cdot \frac{P \cdot R}{P + R},$$

де P – влучність, R – повнота, F – F-міра відповідно, t_p – кількість істинно-позитивних спрацювань, f_p – кількість хибно-позитивних спрацювань, f_n – кількість хибно-негативних спрацювань.

Істинно-позитивними спрацюванням є оцінка, що є позитивною й у тестових даних, і в фінальних результатах роботи методу.

Хибно-позитивними спрацюванням є оцінка, що є негативною й у тестових даних, і в фінальних результатах роботи методу.

Хибно-негативними спрацюванням є оцінка, що є позитивною в тестових даних, але у фінальному результаті роботи методу негативна.

В задачі класифікації оцінка влучності в 1 для класу *C* означає, що кожен зі зразків, який був відмічений методом класифікації як належний до класу *C*, насправді належить до класу *C*. При цьому дана характеристика релевантності не вказує на число зразків з класу *C*, які не було правильно позначено. Повнота в 1 у задачі класифікації означає, що кожен зі зразків, що насправді належіть класу *C*, було позначено як належний до класу *C*. При цьому дана характеристика релевантності не вказує на число зразків з інших класів, що були неправильно позначені як належні до класу *C*.

Між влучністю й повнотою зачасту існує обернена залежність, що приводить до того, що можливо підвищити значення однієї характеристики релевантності зниженням іншої.

Даний компроміс можна розглянути на прикладі наступної ситуації у нейрохірургії. Нейрохірург видаляє ракову пухлину з мозку пацієнта. Нейрохірургові потрібно видалити всі клітини пухлини, оскільки залишені ракові клітини відродять пухлину. І навпаки, нейрохірург мусить не

видаляти здорові клітини мозку, оскільки це призведе до порушень функцій мозку пацієнта. Нейрохірург може бути більш розмашистим щодо області мозку, яку він видаляє, щоби забезпечити видалення всіх ракових клітин. Це рішення підвищує повноту, але знижує влучність. З іншого боку, нейрохірург може бути консервативнішим щодо мозку, який він видаляє, щоби забезпечити вилучення лише ракових клітин. Це рішення підвищує влучність, але знижує повноту. Тобто, вища повнота підвищує шанси видалення здорових клітин (негативний результат), і підвищує шанси видалення всіх ракових клітин (позитивний результат). Вища влучність знижує шанси вилучення здорових клітин (позитивний результат), але також знижує шанси видалення всіх ракових клітин (негативний результат) [21].

Зважаючи на представлену обернену залежність, оцінки влучності та повноти зазвичай не обговорюють окремо, а поєднують в єдину міру. Прикладом такої міри, яка використовується у цій роботі для оцінки методу ідентифікації ризиків і затримок у Agile-проєктах, є F-міра, що розраховується як середнє гармонійне характеристик релевантності влучності та повноти.

Головним недоліком F-міри є те, що при розрахунку середнього гармонійного надається однакова важливість як влучності, так і повноти. При практичному застосуванні, різні типи помилкової класифікації мають різний ефект та призводять до різних втрат [22]. Із цього можна зробити висновок, що відносна важливість влучності та повноти є одним із аспектів задачі.

Для задачі ідентифікації ризиків і затримок влучність і повнота є приблизно рівними за важливістю ознаками.

Влучність є важливою, для вірної ідентифікації завдань, що є ризиками або затримкою в Agile-проєкті, бо невірно виявленні ризики використовуватимуть додаткові ресурси Agile-команди, які могли б бути використані для виконання інших завдань, що може стати причиною для

перевикористання ресурсів, що призведе до невдачі проєкту.

Повнота є важливою, для вірної ідентифікації завдань, що є ризиками або затримкою в Agile-проєкті, бо невиявлені вчасно ризики, можуть призвести до того, що невиявлений ризик приведе до затримки невиявленого завдання та можливий кумулятивний ефект для інших завдань та проєкту в цілому, що призведе до невдачі проєкту.

Зважаючи на це, F-міра є характеристикою релевантності, що може застосовуватись для оцінки методу ідентифікації ризиків та затримок у Agile-проєктах.

Оцінюватись будуть наступні методи машинного навчання, що були розглянуті у розділі 1.2, а саме:

- метод випадкового лісу;
- штучні нейроні мережі;
- наївний Баєсів класифікатор;
- дерево рішень.

Для оцінки ефективності роботи існуючих методів проведемо експеримент. Застосовуючи створений з проблем проєктів з відкритим кодом набір данних, навчимо моделі на основі методів машинного навчання, що були розглянуті у першому розділі. Для оцінки використаємо реалізації методів з бібліотеки *scikit-learn* [23].

Результати оцінки відбирались відповідно до середнього значення для 3 експериментів та відображені у таблиці 2.2, а також на рис. 2.1.

Як можна побачити з результатів проведених експериментів, найкращі результати для задачі ідентифікації ризиків і затримок в Agile-проєктах у порівнянні з іншими методами класифікації дає метод випадкового лісу з досить високими результатами, а саме забезпечується влучність – 0.821, повнота – 0.661, F-міра – 0.732. Також у інших роботах по цій темі метод випадкового лісу показував теж найкращі результати [24].

Таблиця 2.2

Характеристики релевантності існуючих методів

Метод	Влучність	Повнота	F-міра
Наївний Баєсів класифікатор	0,675	0,61	0,64
Дерево рішень	0,786	0,63	0,699
Штучні нейронні мережі	0,77	0,655	0,708
Випадковий ліс	0,821	0,661	0,732

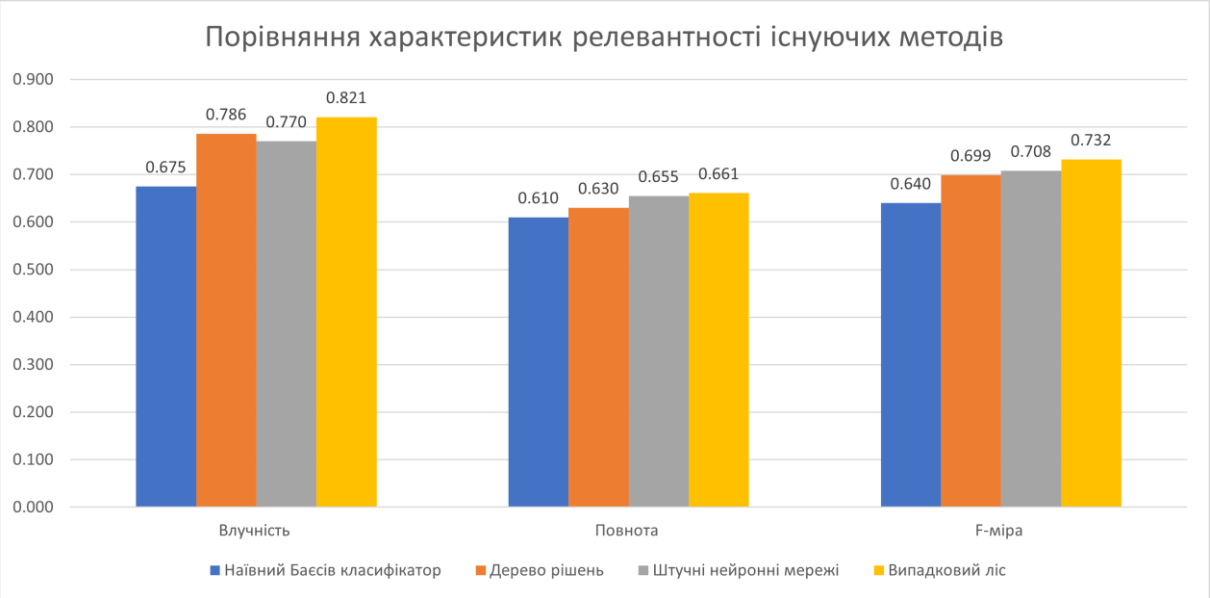


Рис. 2.1. Графік порівняння характеристик релевантності існуючих методів

При цьому недоліком усіх розглянутих методів є відсутність можливості оцінки параметрів у форматі дат, таких як час початку роботи над завданням. Як було зазначено раніше, метод випадкового лісу показав найкращі результати для задачі ідентифікації ризиків та затримок у Agile-проєктах. Але цей алгоритм не включає функції, що пов'язані з датою, які можуть впливати на ймовірність затримки проблеми, наприклад час, коли проблема була подана, або над нею розпочато роботу. Щоб включити функції, пов'язані з датою, можна використати процедуру Prophet [25] і

розрахувати коефіцієнт, який вплине на кінцевий вихід алгоритму випадкового лісу.

Зважаючи на отримані результати, найбільш доцільним для подальшого покращення є метод випадкового лісу з використанням процедури прогнозування Prophet.

2.3. Запропонований метод ідентифікації ризиків і затримок

На основі проведеного аналізу було складено модифікований метод, який дозволяє класифікувати чи є певна задача у методології Agile можливими ризиком. Запропонований метод включає в себе наступні кроки:

1. Збір історичних даних з проєктів та попередня обробка зі зведенням даних до єдиної схеми.
2. Навчання методу випадкового лісу, що використовує локальні характеристики, як пріоритет, тип задачі, час обговорення т.п., що були описані в підрозділі 2.1.
3. Побудова залежності впливу даних на основі дат на ймовірність, що завдання є ризиком, використовуючи процедуру Prophet.
4. Об’єднання результатів навчених моделей в єдиний результат.
5. Використання отриманої об’єднаної моделі для ідентифікації чи є завдання можливим ризиком.

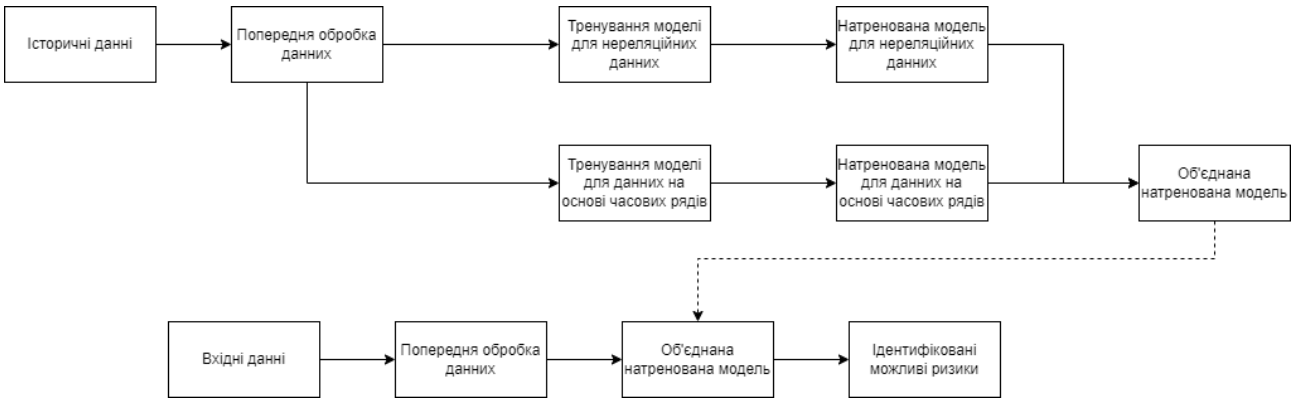


Рис. 2.2. Схема потоку даних методу

На рис. 2.2 подано схему потоку даних методу ідентифікації ризиків

та затримок у Agile-проєктах. На представленій схемі видно вхідні дані та на якому етапі формуються відповідні проміжні дані.

2.4. Опис запропонованого методу

Розглянемо метод випадкового лісу та процедуру Prophet більш детально.

Випадковий ліс

Випадковий ліс прогнозує значення відповіді шляхом усереднення прогнозованих значень відповіді по багатьох деревах рішень [26]. Кожне дерево вирощується на початковому зразку навчальних даних. Завантажувальна вибірка – це випадкова вибірка спостережень, зібрана із заміною. Крім того, предиктори відбираються під час кожного розбиття в дереві рішень. Дерево рішень підбирається за допомогою методології рекурсивного розбиття. Процес підгонки навчального набору відбувається наступним чином:

- для кожного дерева обирається початкова вибірка спостережень;
- встановлюється індивідуальне дерево рішень за допомогою рекурсивного розбиття таким чином:
  - вибирається випадковий набір предикторів для кожного розбиття;
  - розділення продовжується, доки не буде виконано правило зупинки, вказане у вікні специфікації випадкового лісу;
  - кроки 1 і 2 повторюються, доки не буде досягнуто кількості дерев, указаних у вікні специфікації випадкового лісу, або доки не відбудеться рання зупинка.

Отримані дерева рішень потім використовуються для класифікації даних, де результатом випадкового лісу є клас з найбільшою ймовірністю.

Для визначення результату методу випадкового лісу з набору дерев рішень використовувалась наступна функція:

$$P(x) = \frac{\sum_{i=0}^{n} P_i\,(x)}{n},$$

де $P(x)$ – результат класифікації для проблеми x, $P_i\,(x)$ – результат класифікації для проблеми x для піддерева i, n – кількість піддерев в ансамблевому методі.

Prophet

Prophet або «Facebook Prophet» – процедура прогнозування часових рядів на основі адитивної моделі, призначена для обробки генеральних рядів, у яких зручно коригуються нелінійні часові параметри [25]. Правильне прогнозування даних на основі часових рядів допомагає компаніям у плануваннях роботи, постановці цілей і виявленні аномалій. Незважаючи важливість прогнозування, створенням надійних і високоякісних прогнозів є складним процесом, особливо коли є різноманітні часові ряди, а аналітики, які мають досвід моделювання часових рядів, зустрічаються відносно нечасто. Процедура Prophet був розроблений для вирішення даної проблеми та спрощення прогнозування даних часових рядів.

Процедура Prophet має наступні характеристики:

- найкраще працює з часовими рядами, які мають сильний сезонний вплив і кілька періодів історичних даних;
- стійкий до пропущених точок і змін трендів і добре справляється з відхиленнями;
- підтримує тренди, сезонність і свята;
- відповідна бібліотека надає інструменти для автоматичної оцінки та побудови графіків;
- прогноз можна робити автоматично, а потім коригувати вручну;

- результат прогнозування також можна покращити за допомогою інтерпретованих параметрів (знання предметної області та період дослідження).

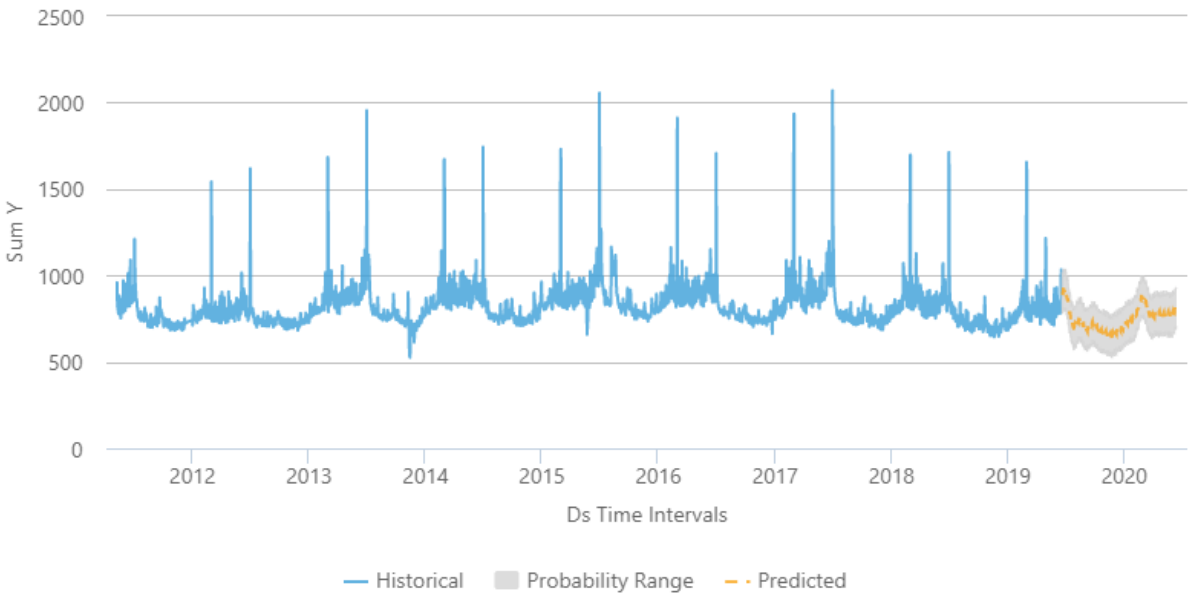


Рис. 2.3. Візуалізація роботи процедури прогнозування Prophet

Щоб включити дані, пов'язані з датою, модель Prophet була навчена передбачати ймовірність того, яка проблема стане ризикованою, якщо робота над проблемою починається у відповідну дату. Середня ймовірність того, що проблема буде затриманою та становитиме ризик, становить приблизно 16,7%, але для проблем, які почалися наприкінці кварталів або під час святкових сезонів, такий шанс вищий, ніж середній наприклад 17,4% для різдвяних свят. Таким чином, можна припустити, що використання коефіцієнта, який дорівнює частці ділення прогнозованої ймовірності ризику моделі Prophet на середню ймовірність, для зміни результатів розподілу в наборах дерева рішень.

Запропонований коефіцієнт розраховується за наступною формулою:

$$k(x) = \frac{k_{prognosed}(x)}{k_{avg}},$$

де $k(x)$ – коефіцієнт розрахований для проблеми x, $k_{prognosed}(x)$ – прогнозована процедурою Prophet ймовірність, що проблема x є ризиком, k_{avg} – середня ймовірність що проблема є можливим ризиком.

Скористаємося отриманим коефіцієнтом для модифікації функції визначення результату методу випадкового лісу з набору піддерев. Отримана функція для запропонованого методу виглядає наступним чином:

$$P(x) = \frac{k_{prognosed}(x)}{k_{avg}} \cdot \frac{\sum_{i=0}^{n} P_i(x)}{n},$$

де $P(x)$ – результат класифікації для проблеми x, $P_i(x)$ – результат класифікації для проблеми x для піддерева i, n – кількість піддерев в ансамблевому методі, $k_{prognosed}(x)$ – прогнозована процедурою Prophet ймовірність, що проблема x є ризиком, k_{avg} – середня ймовірність що проблема є можливим ризиком.

2.5. Висновки до розділу

У рамках цього розділу було досліджено зміст завдання у Agile-проєктах та описано параметри, які є можливими факторами ризику та можуть бути використані у методах машинного навчання для ідентифікації ризиків та затримок. Для визначених параметрів було виконано логістичну регресію для оцінки впливу кожного з запропонованих параметрів проблем на ймовірність, що проблема є ризиком.

Описано характеристики релевантності, які використовувалися для оцінки ефективності роботи існуючих методів. Були оцінені чотири базові методи машинного навчання серед яких метод випадкового лісу, наївний баєсівський класифікатор, дерево рішень та штучні нейронні мережі. Для оцінювання використано набір даних із проєктів з відкритим кодом. Для кожного з цих методів було підраховано значення влучності, повноти та F-міри. У результаті метод випадкового лісу було обрано, як метод з найкращими значеннями.

Запропоновано та описано модифікацію методу випадкового лісу, що враховує дані на основі дат за допомогою процедури Prophet, для вирішення задачі з ідентифікації ризиків і затримок в Agile-проєктах.

3. АНАЛІЗ ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

3.1. Обґрунтування вибору засобів реалізації ПЗ

Для розроблення програмного забезпечення, що реалізує запропонований метод ідентифікації ризиків та затримок у Agile-проєктах, було обрано наступний стек технологій:

1. В якості мови програмування для тренування та тестування моделей машинного навчання використано мову програмування Python.
2. В якості мови програмування для створення та обробки наборів даних використано мову програмування Javascript та фреймворк NodeJs.
3. Для зберігання наборів даних та натренованих моделей було використано нереляційну базу даних MongoDB.
4. Для розробки плагіну для системи відстеження помилок JIRA було використано мову програмування Java, як єдину, що підтримується розробниками системи.

Мова програмування Python

Мова програмування Python – інтерпретована високорівнева мова програмування, з підтримкою значної кількості сучасних парадигм розроблення програмного забезпечення, як об'єктно-орієнтоване, функціональне, процедурне, імперативне та аспектно-орієнтоване програмування [27]. Головною перевагою даної мови є швидкість розроблення ПЗ, що дозволяє значно підвищити продуктивність розробників, за рахунок зручності читання коду та досить простого синтаксису. Для даної мови існує велика колекція бібліотек, що вирішують задачі різного типу.

Мова програмування Python поширюється під вільною ліцензією PSFL, яка дозволяє використовувати дану мову програмування для розробки будь-якого ПЗ. Це дозволяє Python на сьогоднішній день бути

найпопулярнішою мовою програмування у світі та використовуватися у великій кількості компаній з різноплановими проєктами. Мова Python може застосовуватися як основна мова розробки додатків та як мова для розробки розширень програмних застосунків.

Архітектурними рисами мови програмування Python є строга динамічна типізація, багатопоточні обчислення, автоматичне керування пам'яттю, механізм обробки виключень. Також мова Python підтримує механізм розбиття програми на модулі та інтерактивний режим виконання коду. Перевагами мови програмування Python є:

- зрозумілий та простий синтаксис;
- велика колекція багатофункціональних бібліотек;
- кросплатформеність для таких популярних платформ як Microsoft Windows, UNIX, Android та iOS;
- проста інтеграція даної мови у продукти, що використовують інші мови програмування;
- якісна документація до мови програмування Python та супутніх бібліотек.

До недоліків мови програмування Python відносяться:

- відсутність статичної типізації, яка блокує реалізацію механізмів перевантаження функцій;
- неможливість виявлення помилок до виконання ускладнює тестування програм;
- низька швидкість виконання на відміну від застосунків, написаних компільованими мовами.

Java

Java – об'єктно-орієнтована мова програмування, випущена 1995 року компанією «Sun Microsystems» як основний компонент платформи Java [28] Відповідно до індексу TIOBE [29], мова програмування Java є третьою найпопулярнішою з рейтингом 11,74% [30]. Java-програми компілюються у

байт-код, який при виконанні інтерпретується віртуальною машиною для конкретної платформи. Перевагами мови Java є:

- об’єктно-орієнтованість, завдяки чому можливо створювати багаторазові модульні додатки, які легко підтримувати;
- кросплатформеність, для платформ, що спроможні запускати віртуальну машину JVM;
- вища безпека за рахунок функціонування програмних застосунків на мові Java в обмеженому просторі віртуальною машиною JVM;
- вбудована система розподілу стеків дозволяє легко зберігати та отримувати дані в програмних застосунках;
- багатопоточність, що дозволяє програмному застосунку виконуватись у декількох одночасних потоках, що пришвидшує роботу програм;
- велика колекція багатофункціональних бібліотек.

До недоліків мови програмування Java відносяться:

- низька швидкість виконання у порівнянні із застосунками, що запускаються на самій платформі, без необхідності використовувати віртуальну машину;
- високе споживання пам’ять, оскільки для роботи програмних застосунків необхідна віртуальна машина JVM.

Фреймворк NodeJS

JavaScript (JS) – динамічна, об'єктно-орієнтована прототипна мова програмування, реалізація стандарту ECMAScript [31]. Відповідно до індексу TIOBE [29], мова програмування JavaScript є сьомою найпопулярнішою з рейтингом 2,45% [30]. JavaScript підтримує велику кількість сучасних парадигм розробки ПЗ як імперативний, функціональний та прототипний стиль програмування. Мова програмування JavaScript має інтерфейси прикладного програмування для роботи з текстом, датами,

регулярними виразами, стандартними структурами даних та об'єктною моделлю документа (DOM). Перевагами мови JavaScript є:

- висока гнучкість, що дозволяє використовувати різні шаблони програмування залежно від заданої умови;
- підтримка одночасно об'єктно-орієнтованого і функціонального підходів розробки ПЗ;
- кросплатформеність, для таких популярних платформ як Mac OS, UNIX, Microsoft Windows та Android;
- при відсутності критичних помилок, програмне забезпечення на JavaScript буде продовжувати виконувати свої функції, що робить його більш стабільним.

До недоліків мови програмування JavaScript відносяться:

- відсутність багатопоточності, що призводить до виконання коду лише на одному потоці та меншої продуктивності додатку у порівнянні з іншими мовами;
- підтримка лише динамічної типізацію, що робить ускладнює процес знаходження помилок в розроблюваному ПЗ у порівнянні із мовами, що використовують статичну типізацію.

Node.js – це асинхронне, подієве оточення для мови JavaScript. Node.js було створено для побудови масштабних мережевих додатків [32]. Оточення може одночасно обробляти значну кількість з’єднань. Для кожного з’єднання в оточенні викликається функція зворотнього виклику. При відсутності з’єднання, Node.js засинає до створення нового з’єднання. Це відрізняється від загальної моделі обробки з’єднань, що використовує паралельні потоки операційної системи. У Node.js відсутня підтримка багатопоточності, але при цьому можливо використовувати можливості декількох ядер. Також не потрібно турбуватися про блокування процесів, оскільки вони відсутні. Оскільки нічого не блокується, то використовуючи фреймворк Node.js дуже легко створювати масштабовані системи.

Нереляційна база даних MongoDB

MongoDB – це документо-орієнтована СУБД, що широко використовується для роботи із великими об'ємами даних [33]. У нереляційній СУБД дані зберігаються у документах у форматі BSON, який є розширенням класичного JSON та містить набори пар ключ-значення. Перевагами MongoDB є:

- проста структура мови запитів;
- висока швидкість додавання та пошуку даних, коли використовуються правильно обрані індекси;
- широкі можливості вертикального масштабування;
- можливість розміщення кластерів MongoDB на різних хмарних провайдерах, за допомогою сервісу MongoDB Atlas;
- високоякісна система реплікації, що впливає на цілісність документів.

До недоліків СУБД MongoDB відносяться:

- значне використання пам'яті для зберігання даних у порівнянні з іншими СУБД;
- документи, що зберігаються у СУБД не можуть бути більшими за 16 МБ;
- відсутня підтримка транзакцій.

3.2. Опис архітектури розробленого програмного забезпечення

Структурна організація розробленого проєкту зображена нижче на рис. 3.1.

Розроблений програмний застосунок складається із модуля для навчання моделей машинного навчання, модуля ідентифікації ризиків і затримок у Agile-проєктах та плагіна для систему відстеження помилок Atlassian JIRA.

Розглянемо кожен з модулів більш детально.

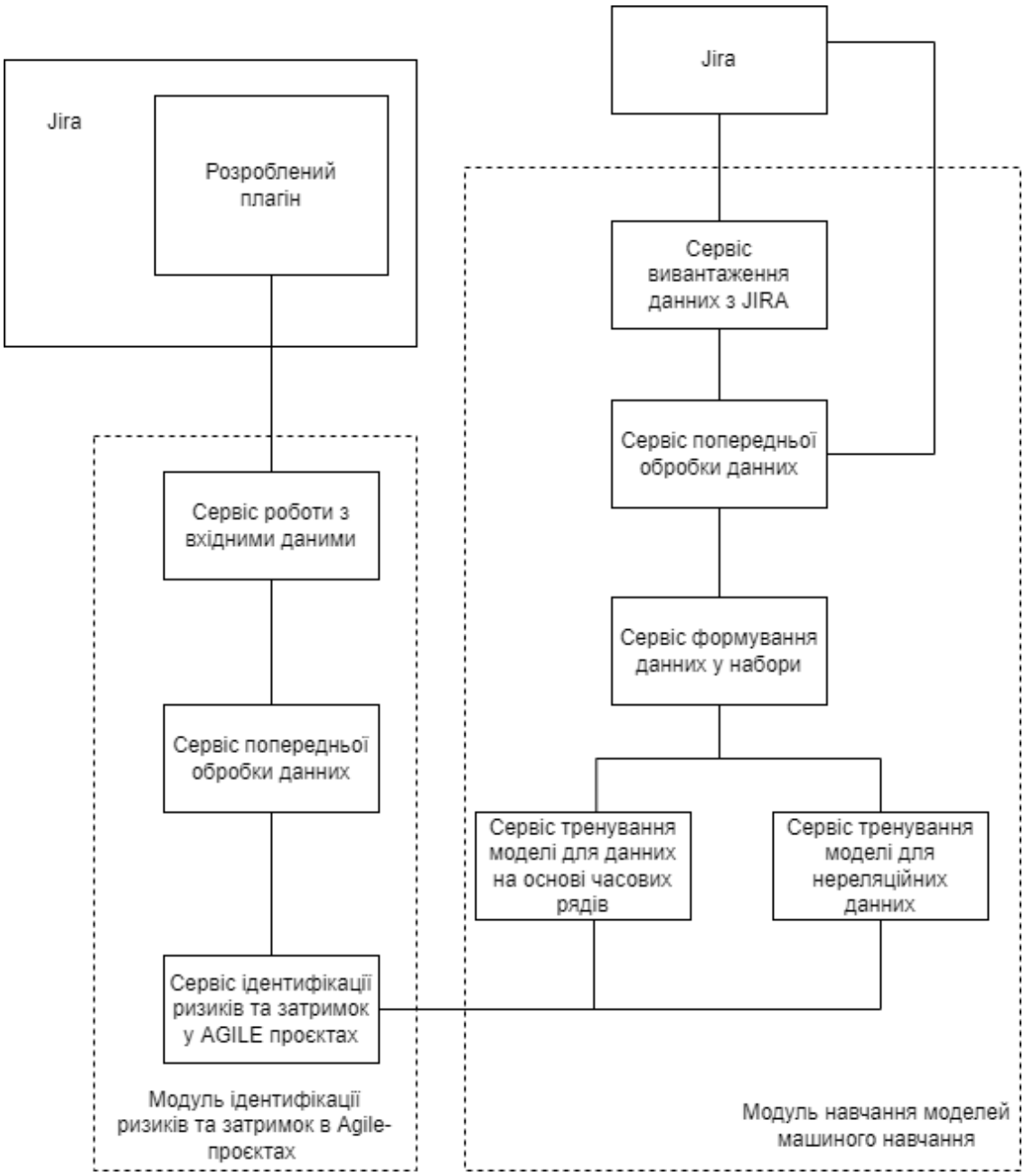


Рис. 3.1. Схема модулів розробленого програмного забезпечення

Для навчання моделей машинного навчання було розроблено модуль навчання моделей, який складається з наступних сервісів:

- сервіс вивантаження даних з системи відстеження помилок JIRA;
- сервіс попередньої обробки даних;
- сервіс формування даних у набори даних;
- сервіс тренування моделі для нереляційних даних;
- сервіс тренування моделі для даних на основі часових рядів.

Схему роботи модуля навчання моделей машинного навчання наведено на рис. 3.2.

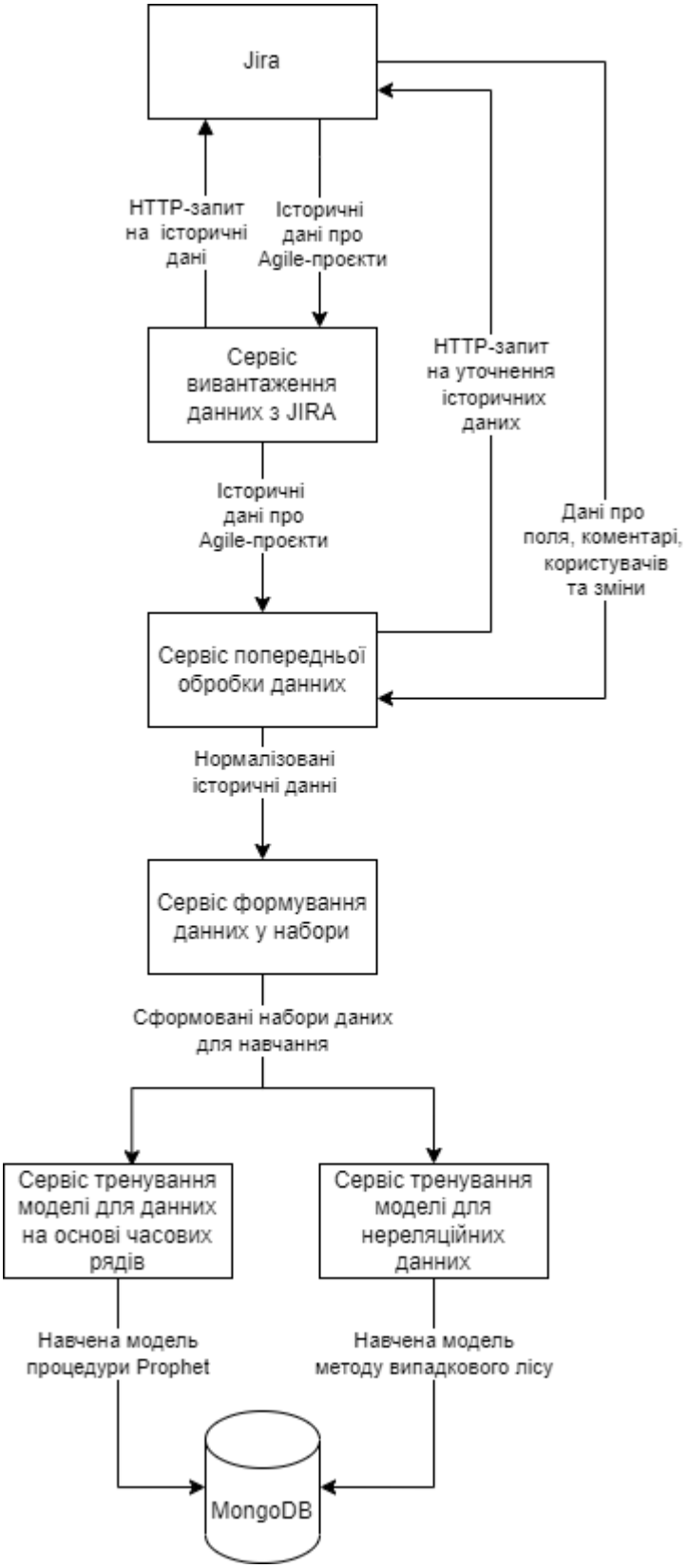


Рис. 3.2. Схема роботи модуля навчання моделей машинного навчання

Для вивантаження історичних даних, що необхідні для навчання моделей машинного навчання, було розроблено сервіс вивантаження даних з системи відстеження помилок JIRA. Даний сервіс використовує вбудований у систему JIRA REST API сервер, який дозволяє вивантажувати дані з системи використовуючи HTTP запити. Дані вивантажуються роблячи HTTP запити на проєкти з відкритим програмним кодом, які використовують систему JIRA для відслідковування проблем, такі як Apache [34], Moodle [35], Red Hat [36], Spring [37], тощо. Вивантаженні дані зберігаються у нереляційній БД MongoDB.

Для попередньої обробки вивантажених з системи JIRA даних було розроблено сервіс попередньої обробки, який приводить отриманні з кількох джерел дані, які мають різну структуру, до єдиної структури, яка використовується для навчання моделей та описана у підрозділі 3.3. Для отримання частини характеристик, як кількість коментарів чи інформації про розробника, що займався певною проблемою, та отримання інформації про значення унікальних полів інших встановлених плагінів, сервіс може робити HTTP-запити на REST API сервер системи відстеження помилок JIRA [38]. Отриманні підготовлені дані зберігаються у нереляційній БД MongoDB.

Сервіс формування даних у набори було розроблено для формування з отриманих даних з різних проєктів, які були зведенні до єдиної структури, у набори даних для тренування та тестування моделей машинного навчання. Розподілені з даних набори зберігаються у нереляційній БД MongoDB.

Для навчання моделі за допомогою методу випадкового лісу використовується сервіс тренування моделі для нереляційних даних, який використовує підготовлені набори даних на основі історичних даних системи JIRA та реалізацію методу випадкового лісу з бібліотеки scikit-learn. Отримана модель зберігається в бінарному форматі в БД MongoDB.

Для навчання моделі за допомогою процедури Prophet використовується сервіс тренування моделі для даних на основі часових

рядів. Отримана модель зберігається у вигляді JSON форматі у БД MongoDB.

Для ідентифікації ризиків та затримок для вхідних даних було розроблено модуль ідентифікації ризиків і затримок у Agile-проєктах, який складається з наступних сервісів:

- сервісі ідентифікації ризиків та затримок у Agile-проєктах;
- сервіс попередньої обробки даних;
- сервіс роботи з вхідними даними.

Схему роботи модуля ідентифікації ризиків і затримок у Agile-проєктах наведено на рис. 3.3.

Отриманні у модулі навчання моделі на основі методу випадкового лісу та процедури Prophet, застосовуються у сервісі ідентифікації ризиків та затримок у Agile-проєктах, який реалізує запропонований у цій роботі метод.

Для роботи з користувачами та Agile командами було розроблено плагін для системи відстеження помилок JIRA, який динамічно підключається безпосередньо до системи JIRA, якою користується Agile команда, та отримує доступ до даних про команду та проблеми і завдання, для яких застосовується метод ідентифікації ризиків і затримок. Для ідентифікації, чи є задача або проблема ризиком, робиться HTTP запит на вебсервер, для якого був розроблений сервіс роботи з вхідними даними, який дозволяє отримувати данні користувачі для ідентифікації ризиків. Отриманні дані передаються у сервіс попередньої обробки, який приводить отриманні з системи відстеження помилок Agile команди дані до структури, яка використовувалась для навчання моделей у запропонованому методі. Для визначення чи є проблеми і завдання зведенні до необхідної структури ризиками, використовується сервіс ідентифікації ризиків та затримок у Agile-проєктах.

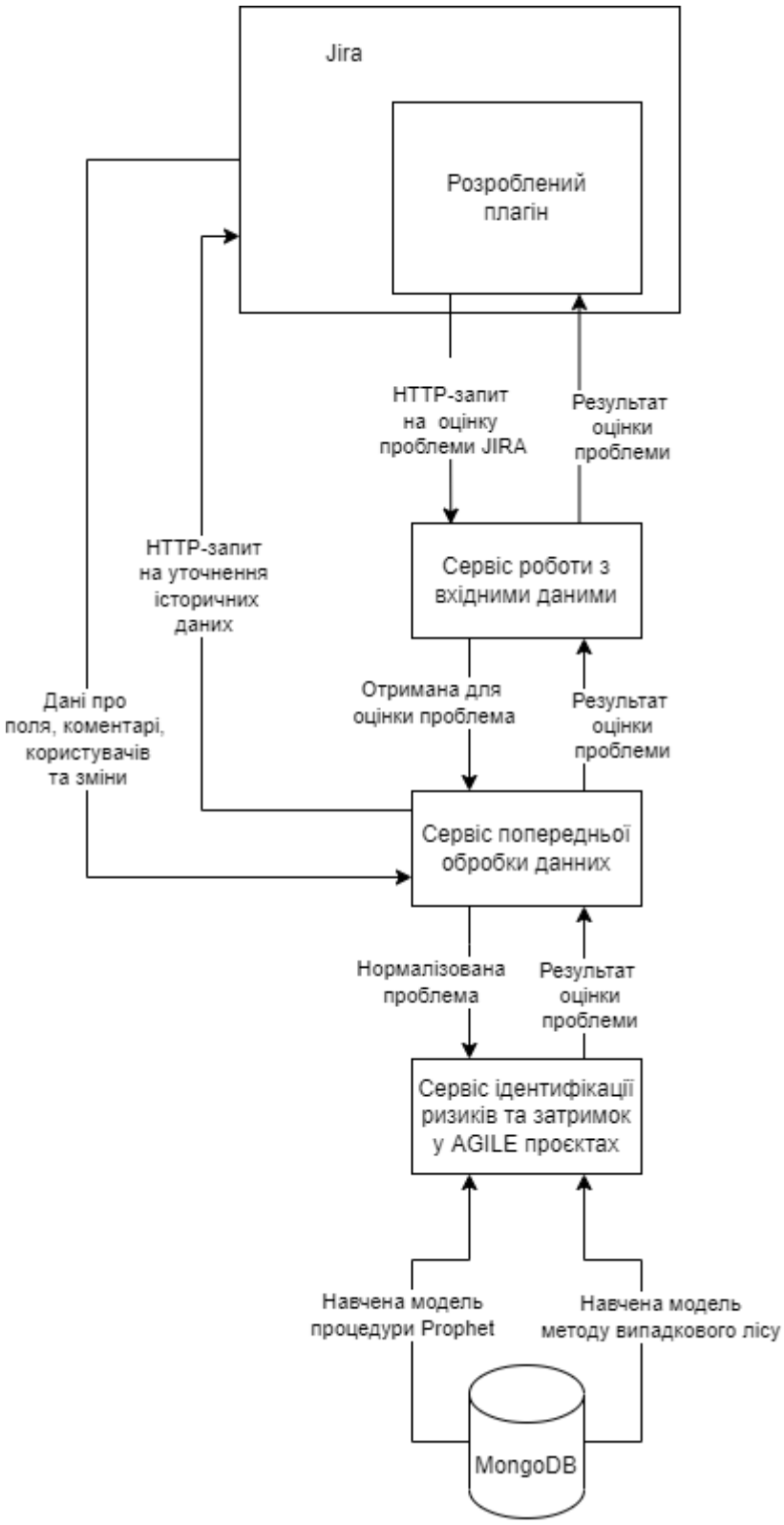


Рис. 3.3. Схема роботи модуля ідентифікації ризиків і затримок у Agile-проєктах

3.3. Структура БД

При розробці системи, було використано нереляційну базу даних MongoDB та були сформовані наступні сутності, що зберігаються у вигляді колекцій документів:

1. Проблема (issue):
    - ідентифікатор файлу (_id) – унікальний ідентифікатор сутності представлений у форматі UUID, автоматично присвоюється усім сутностям у БД MongoDB;
    - ідентифікатор проблеми (id) – унікальний числовий ідентифікатор проблеми, який привласнений системою JIRA;
    - ключ (key) – унікальний текстовий ідентифікатор проблеми, який привласнений системою JIRA;
    - сутність (self) – посилання у URI-форматі на проблему у системі JIRA;
    - поля (fields) – об’єкт, який складається з полів, які зберігають інформацію про відповідну проблему з системи JIRA.
2. Поле (field):
    - ідентифікатор файлу (_id) – унікальний ідентифікатор сутності представлений у форматі UUID, автоматично присвоюється усім сутностям у БД MongoDB;
    - ідентифікатор поля (id) – унікальний текстовий ідентифікатор поля, який привласнений системою JIRA, та застосовується для позначення поля у об’єкті полів у сутності Проблеми;
    - назва (name) – назва поля з якою вона відображається у програмній системі JIRA;
    - схема (schema) – описує тип даних, який зберігається для даного поля у сутності Проблеми.
3. Коментарі (comments):
    - ідентифікатор файлу (_id) – унікальний ідентифікатор сутності представлений у форматі UUID, автоматично присвоюється

усім сутностям у БД MongoDB;

- ідентифікатор проблеми (id) – унікальний числовий ідентифікатор проблеми, до якої належить відповідні коментарі;
- ключ (key) – унікальний текстовий ідентифікатор проблеми, до якої належить відповідні коментарі;
- сума (total) – кількість коментарів, які має відповідна проблема;
- вміст (content) – масив об’єктів, з інформацією про коментарі, які належать до відповідної проблеми системи JIRA.

4. Користувач (user):
    - ідентифікатор файлу (_id) – унікальний ідентифікатор сутності представлений у форматі UUID, автоматично присвоюється усім сутностям у БД MongoDB;
    - ідентифікатор користувача (key) – унікальний текстовий ідентифікатор користувача, який привласнений системою JIRA при створенні користувача;
    - назва (name) – назва поля з якою вона відображається у програмній системі JIRA;
    - флаг активації (active) – показує чи був користувач активованим у системі JIRA;
    - флаг видалення (deleted) – показує чи був користувач видаленим із системи JIRA;
    - часова зона (timezone) – вказує на часову зону, з якої працює користувач;
    - локація (locale) – регіон, з якого працює користувач.
5. Зміна (change)
    - ідентифікатор файлу (_id) – унікальний ідентифікатор сутності представлений у форматі UUID, автоматично присвоюється усім сутностям у БД MongoDB;

- ідентифікатор зміни (id) – унікальний числовий ідентифікатор зміни, який привласнений системою JIRA;
- ідентифікатор проблеми (issue_id) – унікальний числовий ідентифікатор проблеми, до якої належить відповідна зміна;
- автор (author) – автор зміни до проблеми;
- час створення (created) – вказує на час, коли була зроблена зміна до проблеми в системі JIRA;
- елементи (items) – масив, що зберігає інформацію про поля, які були модифіковані у проблемі відповідною зміною.

6. Нормалізована проблема (normalized_issue):
   - ідентифікатор файлу(_id) – унікальний ідентифікатор сутності представлений у форматі UUID, автоматично присвоюється усім сутностям у БД MongoDB;
   - час обговорення (discussion) – цифрове значення, яке вказує на час, який був необхідним для виконання відповідної проблеми;
   - час очікування (waiting) – цифрове значення, яке вказує на час, який був витрачений розробником, який виконує завдання, для зміни статусу її вирішення, як створення пул-реквесту в системі контролю версій;
   - тип (type) – поле, яке позначає тип проблеми, як недолік, завдання, покращення, тощо;
   - кількість перевідкритів (repetition) – кількість разів, коли проблема перевідкривалась;
   - пріоритет (priority) – визначений командою пріоритет проблеми відносно інших проблем команди;
   - кількість зміни пріоритету (no_priority_change) – кількість разів, коли пріоритет проблеми був змінений;
   - кількість коментарів (no_comment) – кількість коментарів, що

члени команди залишили під відповідною проблемою;

- кількість виправлених версій (no_fixversion) – кількість версій ПЗ, для яких виправлена або буде виправлена дана проблема;
- кількість версій з проблемою (no_affectversion) – кількість версій ПЗ, де проблема трапляється;
- кількість пов’язаних проблем (no_issuelink) – кількість проблем, які пов’язані з цією проблемою;
- кількість заблокованих проблем (no_blocking) – кількість проблем, які заблоковані цією проблемою;
- кількість блокувальників (no_blockedby) – кількість проблем, які блокують цю проблему;
- кількість змін опису (no_des_change) – кількість разів опис проблеми був змінений;
- відсоток проблем з затримкою (per_of_delay) – відсоток проблем, якими займався розробник, що були затримані в минулому;
- навантаження (workload) – кількість проблем, якими займається розробник, паралельно з вирішенням даної проблеми;
- репутація репортера (reporter_rep) – репутація репортера, який додав проблему у систему JIRA, вирахувана за відповідною формулою;
- дата початку роботи (work_started_datetime) – дата та час, коли почалась розробка відповідної проблеми;
- флаг ризику (is_risk) – вказує чи є відповідна проблема ризиком.

Описані вище сутності нереляційної БД MongoDB відображені у вигляді ERD-діаграми на рис. 3.4.

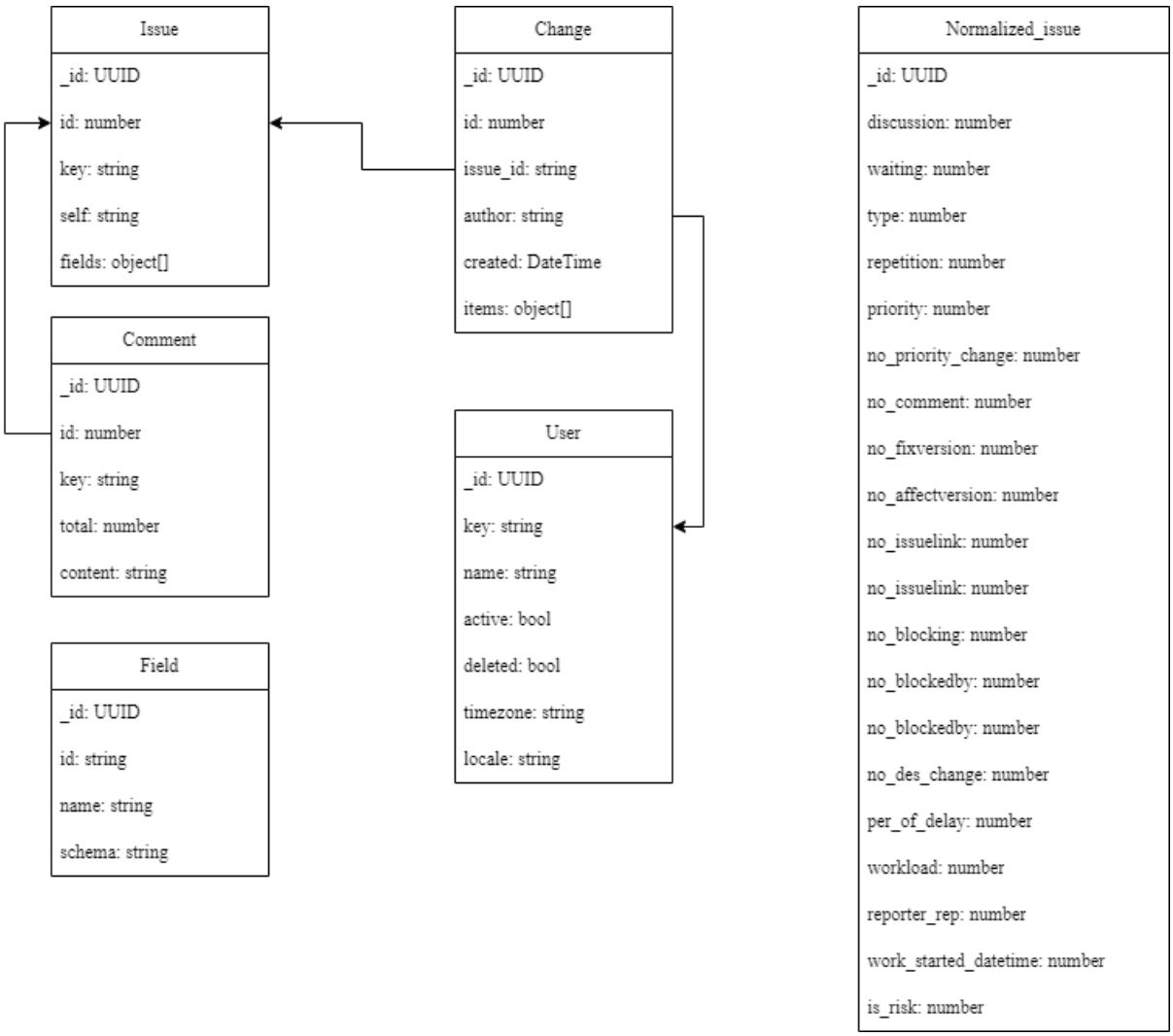


Рис. 3.4. ERD-діаграма сутностей нереляційної БД MongoDB

3.4. Висновки до розділу

У даному розділі було розглянуто засоби реалізації програмного забезпечення, що використовує запропонований метод ідентифікації ризиків та затримок в Agile-проєктах. В якості мови програмування для тренування та тестування моделей машинного навчання було використано мову програмування Python. В якості мови програмування для створення та обробки наборів даних використано мову програмування Javascript та фреймворк NodeJs. Для зберігання наборів даних та натренованих моделей було використано нереляційну базу даних MongoDB. Для розробки плагіну для системи відстеження помилок JIRA було використано мову програмування Java, як єдину, що підтримується розробниками системи.

Було наведено архітектуру розроблюваного програмного забезпечення та описано складові модусі системи. Було описано структуру нереляційної бази даних MongoDB, що була використана в розробленій у цій дисертації програмній системі.

4. АНАЛІЗ ОТРИМАНИХ РЕЗУЛЬТАТІВ

4.1. Підготовка тестового набору даних

Для тестування та оцінки ефективності даної роботи було визначено набір даних і параметрів необхідних для навчання і тестування моделей машинного навчання та додано опис цих параметрів у другому розділі. Для створення потрібного набору даних для навчання та тестування було вирішено зібрати дані проблем з кількох проєктів з відкритим кодом, що застосовують систему Atlassian JIRA, оскільки дана система є однією з найбільш популярних систем, якою користуються Agile-команди, та включає у себе REST API сервер, що дозволяє спростити процес збору даних.

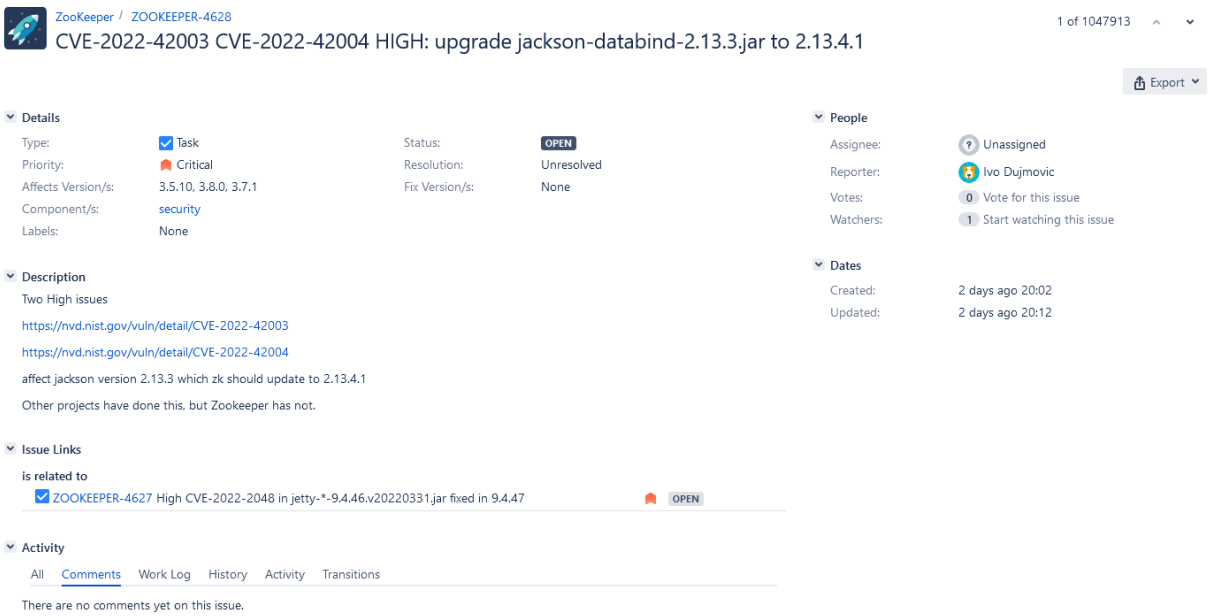


Рис. 4.1. Вигляд задачі у системі JIRA

Під час збору даних, робилися HTTP-запити на сервери системи JIRA, які повертали дані про проблеми, користувачів, поля, коментарі до проблем і історію змін проблем, та отриманні дані зберігалися у БД. Для цього застосовувався програмний сервіс з вивантаження історичних даних, описаний у 3 розділі.

Необхідні данні для навчання та тестування було зібрано з наступних проєктів:

1. Apache – організація-фонд, що сприяє розвитку проєктів програмного забезпечення Apache, як Apache HTTP server, Hadoop, Kafka, Maven, Spark та інші [39]; усього з даного проєкту зібрано більше ніж мільйон проблем для формування набору даних.
2. Moodle – навчальна платформа, призначена для об'єднання педагогів, адміністраторів і учнів в одну надійну, безпечну та інтегровану систему для створення персоналізованого навчального середовища [40]; усього з даного проєкту зібрано близько 60 тисяч проблем для формування набору даних.
3. Red Hat – дистрибутив Linux виробництва компанії Red Hat, орієнтований на комерційний ринок, включно з мейнфреймами. [41]; усього з даного проєкту зібрано близько 375 тисяч проблем для формування набору даних.
4. Spring – фреймворк з відкритим кодом та контейнери з підтримкою інверсії управління для платформи Java [42]; усього з даного проєкту зібрано близько 70 тисяч проблем для формування набору даних.

Усього з 4 проєктів було зібрано близько півтора мільйона проблем.

При цьому для більшості із зібраних проблем немає інформації про зазначений термін виконання або версії релізу, в якій проблема повинна бути вирішена, та дати випуску, яка може використовуватись як термін виконання. Це не дозволяє визначити факт затримки для більшості проблем і такі проблеми не можуть бути використані для навчання та тестування моделей. Зважаючи на умову зазначену вище, набір даних навчання і тестування було сформовано з даних, для яких можна визначити факт ризику, і набір складається з близько 50 тисяч проблем із зібраних проєктів з відкритим кодом, які були приведені до єдиної структури та нормалізовані.

Наступним кроком з підготовки було розбиття отриманого набору на

два піднабори для навчання та тестування. Набір було розділено у співвідношенні 80 та 20 відсотків для навчання та тестування відповідно, при цьому для навчання використовувалися старіші дані, для збереження реалістичного життєвого циклу проєктів, де минулі результати роботи Agile-команд використовуються для визначення майбутніх ризиків.

Детальний розподіл набору на частини для навчання та тестування моделей та відповідно проєктів представлений у таблиці 4.1.

Таблиця 4.1

Розподіл даних з відкритих проєктів у наборі

Проєкт	Набір для навчання			Набір для тестування		
	Ризик	Не ризик	Усього	Ризик	Не ризик	Усього
Apache	1246	5228	6474	461	1615	2076
Moodle	1319	6470	7789	359	1420	1778
Red Hat	3326	16489	19815	812	3326	4138
Spring	539	3839	4378	326	1296	1622
Усього			38456			9614

Усього у сформованому наборі даних було зібрано із проєктів з відритим кодом, що використовують систему відстеження помилок JIRA, 48070 проблем. З них 8714 є затриманими проблемами, що складає 17,4% від усього набору даних.

Отриманий набір було розбито на піднабори для навчання та тестування моделей у відношенні 80 та 20 відсотків, що створило набори розміром у 38456 та 9614 проблем відповідно. При цьому піднабір для навчання моделі застосовує старіші данні, ніж набір для тестування, щоб зберегти реалістичність сценарію ідентифікації ризиків і затримок у Agile-проєктах.

Усього в піднаборі для навчання 6430 завдань, що є ризиком. Це

складає 16,7% від усіх завдань у піднаборі для навчання.

Усього в піднаборі для тестування 1958 завдань, що є ризиком. Це складає 20,3% від усіх завдань у піднаборі для тестування.

Отриманні набори зберігаються у нереляційній БД MongoDB у форматі BSON. Типова проблема, що використовується для навчання та тестування моделей машинного навчання, у форматі BSON зображена на рис. 4.2.


```
{
  "_id": {
    "$oid": "63a17cd5c9d903f63712791e"
  },
  "discussion": 540,
  "waiting": 62,
  "type": 3,
  "repetition": 0,
  "priority": 1,
  "no_priority_change": 1,
  "no_comment": 7,
  "no_fixversion": 2,
  "no_affectversion": 1,
  "no_issuelink": 1,
  "no_blocking": 1,
  "no_blockedby": 0,
  "no_des_change": 8,
  "per_of_delay": 0.3,
  "workload": 2,
  "reporter_rep": 0.28,
  "work_started_datetime": "2021-11-17T16:47:13Z",
  "is_risk": true
}
```


Рис. 4.2. Типова проблема у форматі BSON у СУБД MongoDB

4.2. Оцінка роботи методу

Для оцінки ефективності роботи запропонованого методу проведемо експеримент, де застосовуючи створений з проблем проєктів з відкритим кодом набір даних, навчимо моделі на основі запропонованого методу та методів машинного навчання, що були розглянуті у першому розділі та оцінені у другому, а саме:

- метод випадкового лісу;

- штучні нейроні мережі;
- наївний Баєсів класифікатор;
- дерево рішень.

Для оцінки ефективності будемо використовувати наступні характеристики релевантності: точність, повнота, F-міра, які вираховуються наступним чином:

$$P = \frac{t_p}{t_p + f_p},$$

$$R = \frac{t_p}{t_p + f_n},$$

$$F = 2 \cdot \frac{P \cdot R}{P + R},$$

де P – влучність, R – повнота, F – F-міра відповідно, t_p – кількість істинно-позитивних спрацювань, f_p – кількість хибно-позитивних спрацювань, f_n – кількість хибно-негативних спрацювань.

Результати оцінки відбирались відповідно до середнього значення для 3 експериментів та відображені у таблиці 4.2, а також на рис. 4.3.

Таблиця 4.2

Характеристики релевантності існуючих та запропонованого методів

Метод	Влучність	Повнота	F-міра
Наївний Баєсів класифікатор	0,675	0,61	0,64
Дерево рішень	0,786	0,63	0,699
Штучні нейронні мережі	0,77	0,655	0,708
Випадковий ліс	0,821	0,661	0,732
Запропонований метод	0,829	0,672	0,75

З результатів проведених експериментів можна побачити, що розроблений модифікований підхід дає досить точні результати

ідентифікації ризиків і затримок в Agile-проєктах у порівнянні з іншими методами, а саме забезпечується влучність – 0.829, повнота – 0.672, F-міри – 0.75.

Чим більшим є кожен показник, а саме точність, повнота та F-міра, тим вища ефективність відповідного методу. У нашому випадку запропонований метод показує вищі на 0,01-0,02 від звичайного методу випадкового лісу, що спочатку показував найкращі результати.

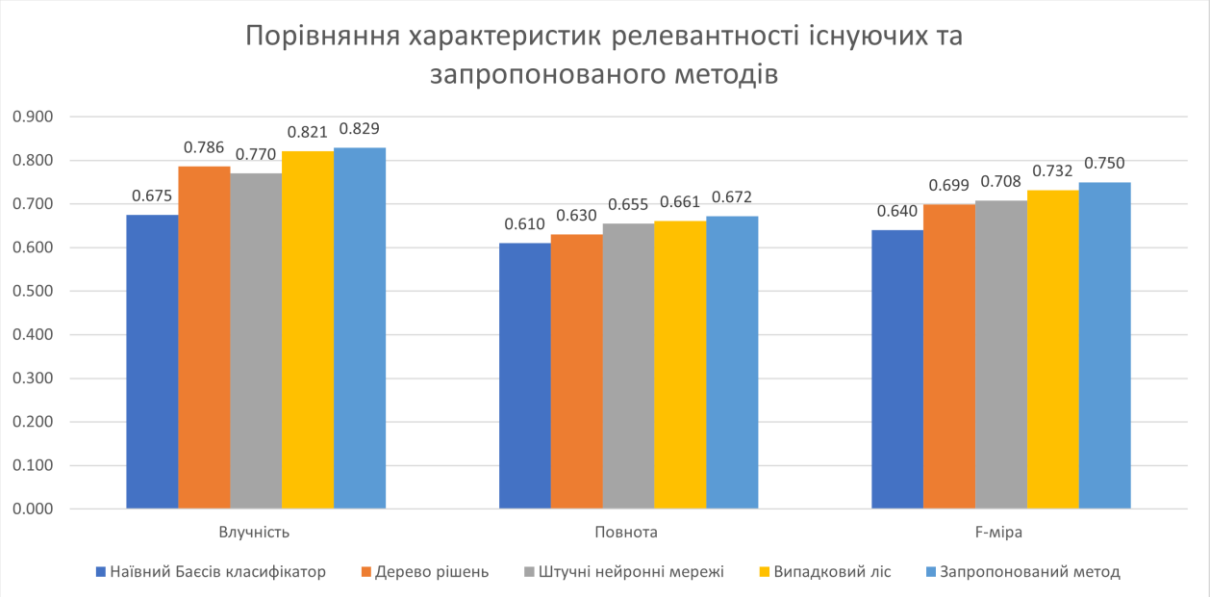


Рис. 4.3. Графік порівняння характеристик релевантності існуючих та запропонованого методів

Зважаючи на це, можна зробити висновок, що данні на основі дат, а саме дата початку роботи над завданнями, мають вплив на задачу з ідентифікації ризиків і затримок у Agile-проєктах, та модифікований метод випадкового лісу з використанням коефіцієнту, який вираховується використовуючи процедуру Prophet.

4.3. Напрямки подальшої роботи

Під час виконання роботи були виявлено наступні напрямки вдосконалення та місця для дослідження.

Удосконалення тестових наборів даних. У даній дисертації для навчання і тестування моделей машинного навчання було використано проблеми з 4 проєктів з відкритим кодом. Для покращення роботи запропонованого методу можна зібрати дані з додаткових проєктів з відкритим кодом.

Розширення факторів ризику командними даними. Перевагою та недоліком параметрів, які використовувалися як фактори ризику в цій роботі, є їхня універсальність, тобто те, що вони можуть використовуватись більшістю Agile-проєктів, але при цьому не враховують додаткових даних про типову команду Agile-проєкту, яка наприклад працює за фреймворком Scrum, тому, що у проєктах з відкритим кодом немає постійної команди. Для врахування даного аспекту необхідно зібрати дані із справжніх комерційних проєктів.

Інтеграція з іншими системами відстеження помилок. Існують інші системи відстеження помилок окрім Atlassian JIRA. Збір даних та створення плагінів для систем, як Azure DevOps [43], є одним з можливих напрямків подальшої роботи.

Семантичний аналіз текстових параметрів завдань. У рамках цієї роботи не проводився аналіз текстових параметрів завдань, як опис або коментарі. Розширення методу за рахунок семантичного аналізу текстових параметрів є одним з можливих напрямків подальшої роботи.

Оцінка впливу пов'язаних завдань. У рамках цієї роботи не проводився аналіз зв'язків між завданнями та вплив цих зв'язків на ймовірність того, чи є завдання ризиком. Розширення методу за рахунок аналізу взаємозв'язків є одним з можливих напрямків подальшої роботи.

4.4. Висновки до розділу

В цьому розділі було описано створені із даних проєктів з відкритим кодом набори даних, які використовувалися для проведення оцінки існуючих та запропонованого методів.

Було проведено оцінку існуючих та запропонованого методів. Запропонований метод показав вищі на 0,01-0,02 характеристики релевантності ніж базовий метод випадкового лісу, що має найкращі результати серед досліджуваних немодифікованих методів.

Також у розділі приведено можливі напрями подальшої роботи над модифікованим методом.

5. ПОБУДОВА БІЗНЕС-МОДЕЛІ

5.1. Опис проблеми

За останні роки неймовірно високим темпом піднімаються прийняття та застосування методології Agile. Станом на 2021 рік, відповідно до опитування “State of Agile”, цією методологією користуються хоча б частково до 95% технологічних компаній [1]. При цьому за рік пандемії частка лише команд розробників ПЗ, що працюються за методологією Agile, збільшилася з 37 відсотків до 86, що більше ніж удвічі. Такі ж показники відносного росту спостерігаються також і для команд, які зазвичай не застосовували методи Agile, як то маркетинг, HR та фінанси. Хоча є багато переваг прийняття цієї методології, також є і вагомі складнощі, що спостерігаються при переході на дану методологію та проявляються при роботі з нею [2].

Однією із труднощів є ідентифікація елементів, які необхідно додавати в Agile беклог. Насамперед основну складність несе велика кількість джерел, звідки ці елементи можуть бути отримані, як запити на нові функції від клієнтів, результати тестувань, знайденні кінцевими користувачами проблеми, результати обговорення Agile команди, тощо. Аналіз отриманої інформації є трудомістким процесом і є вагомим викликом, який при неправильній ідентифікації елементів може привести до фатальних помилок. Після, для уже ідентифікованих елементів беклогу необхідно врахувати залежності з іншими елементами, що для типового проєкту може означати залежність із понад сотнею інших елементів [44].

По-друге, отриманні раніше елементи зазвичай потребують уточнень. Це як і збільшення опису уже існуючих елементів, так і розбиття на менші частини. Особливо важливими є другий, бо зазвичай першочергово утворенні елементи є занадто великими та їх виконання не вписується в один спринт, а отримані результати розбитого елементу усе ще повинні приводити до бажаного результату оригінального елементу.

Ще однією із труднощів, з якою стикаються Agile команди, є

складність планування спринтів. Ключовою частиною цього процесу є обрання групи елементів беклогу, яку планують виконати за спринт. При цьому складно розрахувати таку суму роботи, що може бути виконана із високою ймовірністю, зважаючи на велику кількість факторів, як незаплановані ризики, затримки в інших командах, тимчасова відсутність членів команди [45].

Все вище описане можна узагальнити у схемі «Дерева проблем», яка зображена на рис. 5.1.

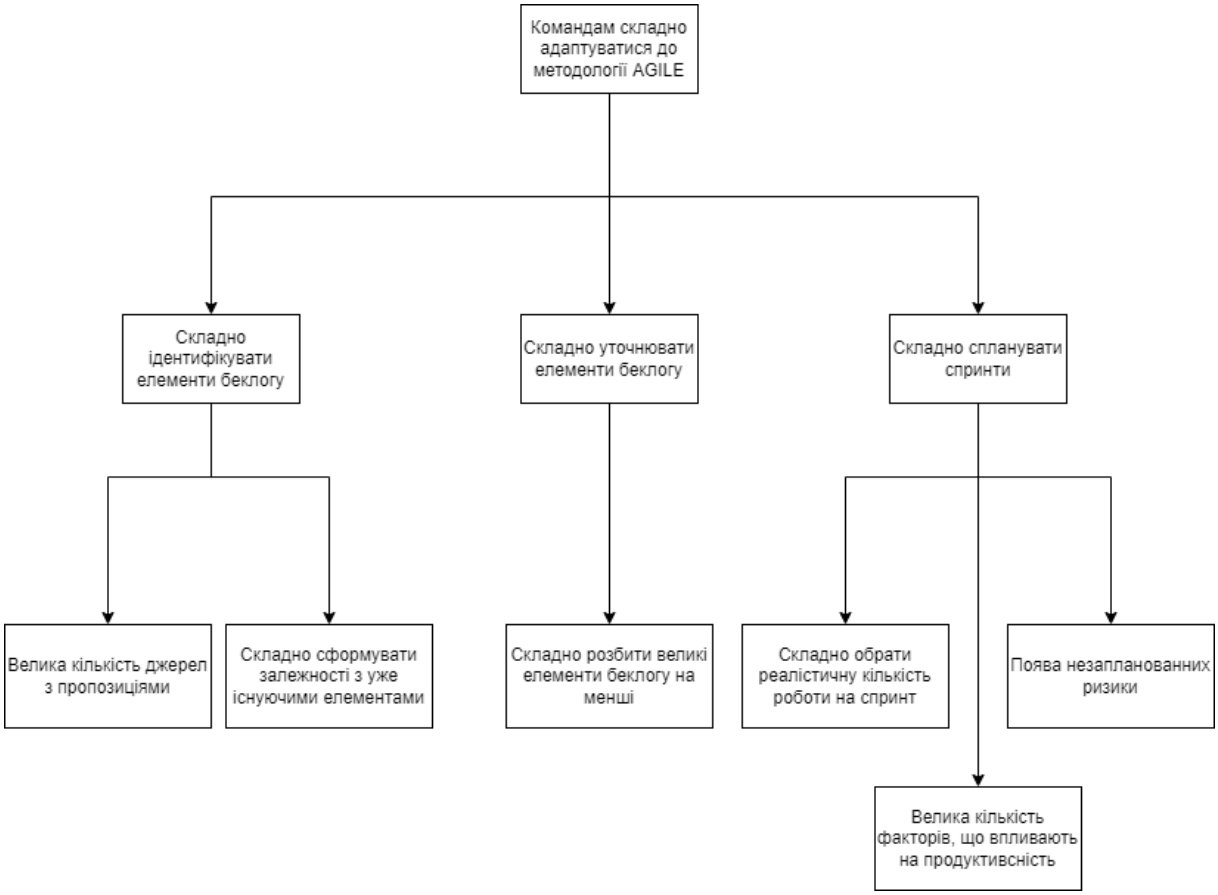


Рис. 5.1. Дерево проблем

5.2. Зацікавлені сторони

У вирішенні описаної вище проблеми існує декілька зацікавлених сторін.

Найбільш очевидною зацікавленою стороною є Agile-команди. Для даних команд, рішення описаної проблеми допоможе зменшити час, який

потрібен для організації процесу роботи Agile команди, за рахунок зменшення трудомісткості даних процесів. Також дана система дозволить Agile-командам краще управляти командними ресурсами, уникати зайвих затримок та в цілому допоможе збільшити ймовірність успіху окремо взятих проєктів.

Другою зацікавленою стороною виступають замовники проєктів. За рахунок вирішення висвітлених проблем, досягається більш ефективна робота над проєктом та швидше отримання результату. Це означає, що для досягнення успіху окремо взятого проєкту, замовникам необхідно буде витрачати менш ресурсів для досягнення необхідного результату. Сторона замовника зацікавлена в отриманні результату, використовуючи меншу кількість ресурсів.

У табл. 5.1 зведено усі групи зацікавлених сторін, їх інтереси, та вплив.

Таблиця 5.1

Зацікавлені сторони

Зацікавлена сторона	Інтерес зацікавленої особи	Вплив зацікавленої особи	Стратегії приваблення зацікавлених сторін
Члени Agile команд	Простіший та менш трудомісткий процес організації роботи команди	Високий	Проведення презентації для представників зацікавлених осіб. Участь у спеціалізованих виставках та форумах.
Замовники проєктів	Більш ефективна робота над проєктом та швидше отримання результату	Високий	

5.3. Комерційне рішення. Основні характеристики

Сформуємо вимоги до комерційного рішення на основі приведеного вище дерева проблем:

- аналіз елементів, які пропонується додати у беклог, та знаходження залежностей між ними та вже існуючими елементами;
- визначення можливих методів розбиття існуючих елементів на під елементи;
- визначення групи елементів беклогу, які можливо взяти у спринт, спираючись на історичні тренди та можливості команди;
- ідентифікація можливих ризиків та затримок серед елементів беклогу.

Розроблений продукт представляє собою комплекс програмних модулів, що реалізує програмний метод ідентифікації ризиків та затримок у Agile-проєктах, описаний у попередніх розділах. Такий плагін є інтегрованим з існуючими інструментами для управління Agile-проєктами, а саме Atlassian JIRA – найбільш популярною системою відстежування помилок.

5.4. Конкурентні переваги рішення

На сьогоднішній день сфера з вирішення описаної проблеми є малою, та саме програмна реалізація, яка б могла повноцінно вирішувати дану проблему не існує, а більшість програмних систем налаштовані на пошуки ризиків, а не спрощення розробки. При цьому для рішення даних проблем деякі команди наймають спеціалістів, що займаються лише питанням організації, а не розробки. При використанні створеного метода є можливість зменшити кількість працезатрат на виконання аналізу й організацію робочого процесу та оптимізувати задачі і ролі членів Agile-команд.

Також серед існуючих рішень, є жорстке прив'язування до відповідної

системи організації роботи, що обмежує гнучкість команди, і підтримка модулем кількох різних систем відстежування помилок, як Attlassian JIRA, Azure DevOps, Digital.ai, може бути конкурентною перевагою, особливо при пропозиції даного рішення великим компаніям, що можуть застосовувати у своїх робочих процесах декілька різних, незалежних одна від одної систем відстежування помилок.

Отже, використання програмного методу побудови рекомендацій для системи організації командної роботи, описаної в дослідженні, могло б кількість працезатрат на виконання аналізу й організації робочого процесу та оптимізувати задачі і ролі членів Agile-команд, особливо для тих команд, що використовують декілька систем організації командної роботи одночасно, за рахунок розробки плагінів для таких систем, що стає можливим завдяки відсутності залежності розробленого методу від ознак лише однієї системи.

5.5. Клієнти. Сегменти ринку споживання

Розроблений продукт найбільше підходить для реалізації на ринку b2b. Основними потенційними клієнтами продукту є компанії, що ведуть розробку програмного забезпечення та діяльність своїх команд за методологією Agile.

Продуктом можуть зацікавитись компанії, що ведуть проєкти за методологією Agile, які за рахунок використання продукту зможуть зменшити кількість працезатрат на виконання аналізу й організації робочого процесу та оптимізувати задачі і ролі членів Agile команд, та також за рахунок підтримки уже існуючих систем організації роботи, зможуть інтегрувати продукт у повсякденне користування без необхідності зміни програм, якими користувалися раніше. Сюди також можна віднести компанії, що лише планують переходити на використання методів Agile і для яких застосування методів спростить перехід.

Крім того, продукт може бути реалізований і для ринку b2c. Клієнтами

можуть бути розробники, які використовують методологію Agile для власних невеликих проєктів з малими командами ентузіастів. Однак, оскільки проєкти малі, вони мають зазвичай недостатньо значний беклог та меншу необхідність в управлінні ресурсами, що робить даний сегмент не перспективним для розробленого програмного рішення.

5.6. Унікальна ціннісна пропозиція

У дереві проблем було виділено, яке саме завдання даний програмний продукт має вирішити, а в описі зацікавлених стороні було визначено, які саме очікування від продукту наявні у потенційних користувачів.

Компанії з розробки програмного забезпечення, що у більшості своїй використовують методологію Agile [1], зацікавленні у спрощенні процесу організації роботи Agile-команди та зменшені працезатрат та витрат інших ресурсів на таку організацію. У той час замовники Agile-проєктів зацікавлені у більш ефективній роботі над проєктом, ефективній витраті ресурсів та швидшому отриманні результатів від Agile-проєкту.

Запропонований програмний метод ідентифікації ризиків та затримок в Agile-проєктах та розроблені на основі запропонованого методу плагіни для популярних систем організації командної роботи та відстеження помилок, як Atllasian JIRA, Azure DevOps, Digital.ai, відповідає таким вимогам, вирішує проблеми та задовольняє очікування цільових користувачів продукту, компанії з розробки програмного забезпечення, що використовують методологію Agile.

Отже, унікальною ціннісною пропозицією є розроблений програмний методу ідентифікації ризиків та затримок в Agile-проєктах та розроблені на його основі плагіни для систем організації командної роботи, що зменшує працезатрати на організацію командної роботи.

5.7. Доходи та витрати

Дохід складається з продажів ліцензії на користування створеним

програмним методом.

Доцільне створення декількох типів ліцензій, що покривають різні сценарії використання розробленого плагіну.

Наприклад ліцензія для комерційного використання програмного забезпечення, при купівлі якої користувач отримає обмежені права використання програмного забезпечення в комерційних цілях. При цьому забороняється вивчення, внесення змін, тиражування, розповсюдження та перепродаж програмного коду або його частин. Технічна підтримка передбачає зворотній зв’язок із розробником з подачею скарг на некоректну роботу програмного забезпечення, та доступ до оновлень.

Таблиця 5.2

Витрати на реалізацію проєкту

Найменування витрат	1-й місяць, т. $	2-й місяць, т. $	3-й місяць, т. $	4-й місяць, т. $	5-й місяць, т. $	6-й місяць, т. $
Загальні витрати	2	2	2	2	2	2
ЗП	–	25	25	25	25	25
Витрати	2	27	27	27	27	27
Заплановані прибутки	–	–	–	–	–	–
Результат без оподаткування:	–2	–27	–27	–27	–27	–27

Також можна запропонувати ліцензію для індивідуальних користувачів та малих Agile-команд. Вона може розповсюджуватись безкоштовно, але при цьому данні, які збираються з систем організації командної роботи Agile-проєктів, що використовують даний тип ліцензії обов’язково передаються розробникам та можуть застосовуватись для

покращення запропонованого в цій роботі методу. Для користувачів комерційної ліцензії передача даних не є обов'язковою. При цьому забороняється вивчення, внесення змін, тиражування, розповсюдження та перепродаж програмного коду або його частин. Технічна підтримка не передбачає зворотній зв'язок із розробником з подачею скарг на некоректну роботу програмного забезпечення, але надає доступ до оновлень. Також для користувачів, що використовують персональну ліцензію, непередбачене використання програмного продукту у комерційних цілях.

Визначимо основні статті витрат:

- юридичне консультування;
- послуги з ведення рахунків та бухгалтерії;
- оформлення патентних документів;
- робота команди розробників;
- робота служби підтримки;
- сплата податків;
- маркетинг, реклама, розповсюдження продукту.

Продовження табл. 5.2

Найменування витрат	7-й місяць, т. $	8-й місяць, т. $	9-й місяць, т. $	10-й місяць, т. $	11-й місяць, т. $	12-й місяць, т. $
Загальні витрати	2	2	2	2	2	2
ЗП	28	28	28	28	28	28
Витрати	30	30	30	30	30	30
Заплановані прибутки	75	75	75	75	75	75
Результат без оподаткування:	45	45	45	45	45	45

Доходи на витрати на кожен місяць описано в таблиці 5.2.

5.8. Бізнес-модель

Узагальнимо, все написане вище у лаконічну бізнес-модель у вигляді lean canvas.

Таблиця 5.3

Канвас бізнес-моделі

<table>
<tr>
<td rowspan="2">Проблема

складність ідентифікації елементів беклогу,

складність уточнення елементів беклогу,

складність планування спринтів</td>
<td>Рішення

програмний метод побудови рекомендацій для системи організації командної роботи;</td>
<td rowspan="2">Унікальна ціннісна пропозиція

програмний метод побудови рекомендацій для системи організації командної роботи;

зменшення працезатрат на організацію командної роботи</td>
<td>Прихована перевага

можливість інтегрування програмного продукту з будь-якими великими системами, що спеціалізуються на організації командної роботи</td>
<td rowspan="2">Споживачі

компанії, що ведуть проєкти за методологією Agile або планують її застосовувати</td>
</tr>
<tr>
<td>Ключові метрики

кількість проданих ліцензій</td>
<td>Канали

через відділи співпраці/ інтеграції відповідних соціальних мереж</td>
</tr>
<tr>
<td colspan="3">Структура витрат

утримання команд розробки та технічної підтримки; оплата послуг юриста, бухгалтера; маркетинг</td>
<td colspan="2">Потоки доходів

доходи від продажу ліцензій; доходи від підтримки програмного забезпечення</td>
</tr>
</table>

Споживачі: компанії, що використовують методологію Agile або планують її застосовувати.

Проблема: складність ідентифікації елементів беклогу, складність уточнення елементів беклогу, складність планування спринтів, для проєктів з великими розмірами беклогів.

Рішення: плагіни для найпопулярніших системи організації командної роботи як Atllasian JIRA, Azure DevOps, Digital.ai, що використовують запропонований у цій роботі програмний метод ідентифікації ризиків та затримок в Agile-проєктах.

Унікальна ціннісна пропозиція: програмний метод ідентифікації ризиків та затримок в Agile-проєктах; зменшення працезатрат на організацію командної роботи та ресурсів необхідних для ефективного управління Agile-проєктом.

Потоки доходів: доходи від продажу ліцензій на плагін для систем Attlassian Jira, Azure DevOps, Digital.ai; доходи від підтримки програмного забезпечення.

Структура витрат:

- юридичне консультування;
- послуги з ведення рахунків та бухгалтерії;
- оформлення патентних документів;
- робота команди розробників;
- робота служби підтримки;
- сплата податків;
- маркетинг, реклама, розповсюдження продукту.

Також в канву бізнес-моделі включаються структурні блоки: прихована перевага, ключові метрики та канали.

Канали: через відділи співпраці/інтеграції відповідних соціальних мереж.

Ключові метрики: кількість проданих ліцензій на користування розробленими плагінами, кількість встановлень плагіну клієнтами за

відкритою ліцензією.

Прихована перевага: можливість інтегрування програмного продукту з будь-якими великими системами, що спеціалізуються на організації командної роботи, як Attlassian Jira, Azure DevOps, Digital.ai.

Бізнес-модель наведена у зведеному вигляді у таблиці 5.3.

Отже, зважаючи на дані у таблиці 5.3, можна зробити висновок, що запропонований проєкт, який реалізує описаний у дисертацій метод побудови рекомендацій для системи організації командної роботи, має перспективи у своїй подальшій реалізації, оскільки він має потенційних споживачів, для яких вирішується проблема з ідентифікації ризиків та затримок у Agile-проєктах. Запропоноване рішення має унікальну ціннісну пропозицію для перспективних користувачів та приховану перевагу у порівнянні з конкурентами. Проведений аналіз не враховує всіх ризиків та факторів, як специфіка оподаткування у країні ведення бізнесу, проте навіть зробленого дослідження достатньо для прогнозування комерційного успіху продукту та його окупаємості.

5.9. Висновок до розділу

У даному розділі було проведено аналіз поточної ситуації у сфері побудови рекомендацій для системи організації командної роботи, виявлено наявні проблеми та зображено їх у дереві проблем. Виділено основні зацікавлені сторони у вирішенні існуючих недоліків та ступінь впливу виявлених сторін на вирішення проблем. Для вирішення виявлених проблем запропоновано комерційне рішення з конкурентними перевагами, що задовольняє інтереси розглянутих зацікавлених осіб. Виділено унікальну ціннісну пропозицію запропонованого продукту. Проведено аналіз майбутніх клієнтів та досліджено сегменти ринку споживання, що дозволило спрогнозувати потенційні доходи та витрати на реалізацію продукту з ідентифікації ризиків і затримок у Agile-проєктах. У результаті була описана бізнес-модель, що обґрунтовує доцільність реалізації плагіну,

що застосовує запропонований метод, та прогнозує його потенційну окупаємість та прибутковість.

ВИСНОВКИ

На основі проведеного в даній магістерській дисертації дослідження можна зробити наступні висновки:

1. Описано задачу ідентифікації ризиків і затримок у Agile-проєктах, яка є задачею бінарної класифікації та вирішується методами машинного навчання. Розглянуто та описано чотири базові методи машинного навчання: метод випадкового лісу, наївний баєсівський класифікатор, дерево рішень та штучні нейронні мережі.
2. Досліджено зміст завдання у Agile-проєктах та описано параметри, які є можливими факторами ризику та можуть бути використані для ідентифікації ризиків та затримок. Оцінено чотири базових метода машинного навчання та запропоновано й описано модифікацію для методу випадкового лісу, що показував найвищі значення релевантності, використовуючи процедуру Prophet.
3. Розглянуто засоби реалізації програмного забезпечення, що використовує запропонований метод ідентифікації ризиків та затримок в Agile-проєктах. Наведено архітектуру розроблюваного програмного забезпечення та описано складові модулі системи.
4. Зібрано з проєктів з відкритим програмним кодом та описано набори даних, які використовувалися для проведення оцінки існуючих та запропонованого методів, проведено оцінку існуючих та запропонованого методів. Запропонований метод показав на 0,01-0,02 характеристики релевантності вищі ніж базовий метод випадкового лісу, що має найкращі результати серед досліджуваних немодифікованих методів.

Подальшими напрямами роботи є удосконалення тестових наборів даних, розширення факторів ризику командними даними, інтеграція з іншими популярними системами відстеження помилок, семантичний аналіз

текстових параметрів завдань та оцінка впливу між пов’язаними завданнями.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. State of Agile report [Електронний ресурс]. — Режим доступу: https://stateofAgile.com/.
2. 9 Reasons for Agile Failures [Електронний ресурс]. — Режим доступу: https://zegal.com/blog/post/Agile-methodology-failures/.
3. Har-Peled, S., Roth, D., Zimak, D. (2003) "Constraint Classification for Multiclass Classification and Ranking." In: Becker, B., Thrun, S., Obermayer, K. (Eds) Advances in Neural Information Processing Systems 15: Proceedings of the 2002 Conference, MIT Press. ISBN 0-262-02550-7
4. Stuart J. Russell, Peter Norvig (2010) Artificial Intelligence: A Modern Approach, Third Edition, Prentice Hall ISBN 9780136042594
5. Quinlan, J. R. (1986). "Induction of decision trees" (PDF). Machine Learning. 1: 81–106. doi:10.1007/BF00116251. S2CID 189902138.
6. Ho, Tin Kam (1995). Random Decision Forests (PDF). Proceedings of the 3rd International Conference on Document Analysis and Recognition, Montreal, QC, 14–16 August 1995. pp. 278–282
7. Haykin (2008) Neural Networks and Learning Machines, 3rd edition
8. McCallum, Andrew. "Graphical Models, Lecture2: Bayesian Network Representation"
9. A. Pika, W. M. van der Aalst, C. J. Fidge, A. H. ter Hofstede, M. T. Wynn, and W. V. D. Aalst, “Profiling event logs to configure risk indicators for process delays,” Advanced Information Systems Engineering (CAISE 2013), pp. 465–481, Jul. 2013.
10. T. Zimmermann, N. Nagappan, P. J. Guo, and B. Murphy, “Characterizing and predicting which bugs get reopened,” in 34th International Conference on Software Engineering (ICSE), 2012. IEEE Press, Jun. 2012, pp. 1074–1083
11. H. Valdivia Garcia, E. Shihab, and H. V. Garcia, “Characterizing and predicting blocking bugs in open source projects,” in Proceedings of the

11th Working Conference on Mining Software Repositories - MSR 2014. ACM Press, May 2014, pp. 72–81.

12. N. Bettenburg, S. Just, A. Schroter, C. Weiss, R. Premraj, and¨ T. Zimmermann, “What makes a good bug report?” in Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering - SIGSOFT ’08/FSE-16. New York, New York, USA: ACM Press, Nov. 2008, p. 308. [Online]. Available:http://dl.acm.org/citation.cfm?id=1453101.1453146
13. L. D. Panjer, “Predicting Eclipse Bug Lifetimes,” in Fourth International Workshop on Mining Software Repositories (MSR’07:ICSE Workshops 2007). IEEE, May 2007, pp. 29–29.
14. P. Hooimeijer and W. Weimer, “Modeling bug report quality,” in Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering - ASE ’07. ACM Press, Nov. 2007, p. 34. [Online]. Available: http://dl.acm.org/citation.cfm?id=1321631. 1321639
15. P. J. Guo, T. Zimmermann, N. Nagappan, and B. Murphy, “Characterizing and predicting which bugs get fixed: an empirical study of Microsoft Windows,” 2010 ACM/IEEE 32nd International Conference on Software Engineering, vol. 1, pp. 495–504, 2010.
16. W.-M. Han and S.-J. Huang, “An empirical analysis of risk components and performance on software projects,” Journal of Systems and Software, vol. 80, no. 1, pp. 42–50, Jan. 2007.
17. A. A. Porter, H. P. Siy, and L. G. Votta, “Understanding the effects of developer activities on inspection interval,” in Proceedings of the 19th international conference on Software engineering - ICSE ’97. ACM Press, May 1997, pp. 128–138.
18. L. Wallace and M. Keil, “Software project risks and their effect on outcomes,” Communications of the ACM, vol. 47, no. 4, pp. 68–73, Apr. 2004.

19. S. Kim, T. Zimmermann, K. Pan, and E. Jr. Whitehead, “Automatic Identification of Bug-Introducing Changes,” in 21st IEEE/ACM International Conference on Automated Software Engineering (ASE’06). IEEE, Sep. 2006, pp. 81–90. [Online]. Available: http://dl.acm.org/citation.cfm?id=1169218.1169308
20. Powers, David M. W. Evaluation: From Precision, Recall and F-Measure to ROC, Informedness, Markedness & Correlation // Journal of Machine Learning Technologies, 2011 – P. 37–63.
21. Ting, Kai Ming (2011). Sammut, Claude; Webb, Geoffrey I. (eds.). Encyclopedia of machine learning. Springer. doi:10.1007/978-0-387-30164-8. ISBN 978-0-387-30164-8.
22. Hand, David. "A note on using the F-measure for evaluating record linkage algorithms - Dimensions". app.dimensions.ai. doi:10.1007/s11222-017-9746-6. hdl:*10044/1/46235*. S2CID 38782128.
23. Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.
24. Choetkiertikul, M., Dam, H.K., Tran, T. et al. Predicting the delay of issues with due dates in software projects // Empirical Software Engineering. Vol. 22. 2017. - P. 1223–1263. https://doi.org/10.1007/s10664-016-9496-7.
25. Taylor SJ, Letham B. 2017. Forecasting at scale. PeerJ Preprints 5:e3190v2 https://doi.org/10.7287/peerj.preprints.3190v2.
26. Breiman L. Random Forests // Machine Learning. Vol. 45. 2001. P. 5–32. doi:10.1023/A:1010933404324.
27. Python [Електронний ресурс]. — Режим доступу https://en.wikipedia.org/wiki/Python_(programming_language)
28. Java [Електронний ресурс]. — Режим доступу: https://uk.wikipedia.org/wiki/Java
29. TIOBE Programming Community Index Definition [Електронний ресурс]. — Режим доступу: https://www.tiobe.com/tiobe-index/programming-languages-definition

30. TIOBE Index [Електронний ресурс]. — Режим доступу: https://www.tiobe.com/tiobe-index
31. JavaScript [Електронний ресурс]. — Режим доступу: https://uk.wikipedia.org/wiki/JavaScript
32. Node.js [Електронний ресурс]. — Режим доступу: https://en.wikipedia.org/wiki/Node.js
33. MongoDB [Електронний ресурс]. — Режим доступу: https://uk.wikipedia.org/wiki/MongoDB
34. Apache issues [Електронний ресурс]. — Режим доступу: https://issues.apache.org/jira
35. Moodle issues [Електронний ресурс]. — Режим доступу: https://tracker.moodle.org.
36. Red Hat issues [Електронний ресурс]. — Режим доступу: https://issues.redhat.com.
37. Spring issues [Електронний ресурс]. — Режим доступу: https://jira.spring.io.
38. Attlassian JIRA REST API v2 [Електронний ресурс]. — Режим доступу: https://developer.atlassian.com/cloud/jira/platform/rest/v2/intro.
39. The Apache Software Foundation [Електронний ресурс]. — Режим доступу: https://en.wikipedia.org/wiki/The_Apache_Software_Foundation.
40. Moodle [Електронний ресурс]. — Режим доступу: https://en.wikipedia.org/wiki/Moodle.
41. Red Hat [Електронний ресурс]. — Режим доступу: https://en.wikipedia.org/wiki/Red_Hat.
42. Spring Framework [Електронний ресурс]. — Режим доступу: https://en.wikipedia.org/wiki/Spring_Framework.
43. Azure DevOps [Електронний ресурс]. — Режим доступу: https://azure.microsoft.com/en-us/services/devops.
44. M. Cohn, Agile estimating and planning. Pearson Education, 2005.

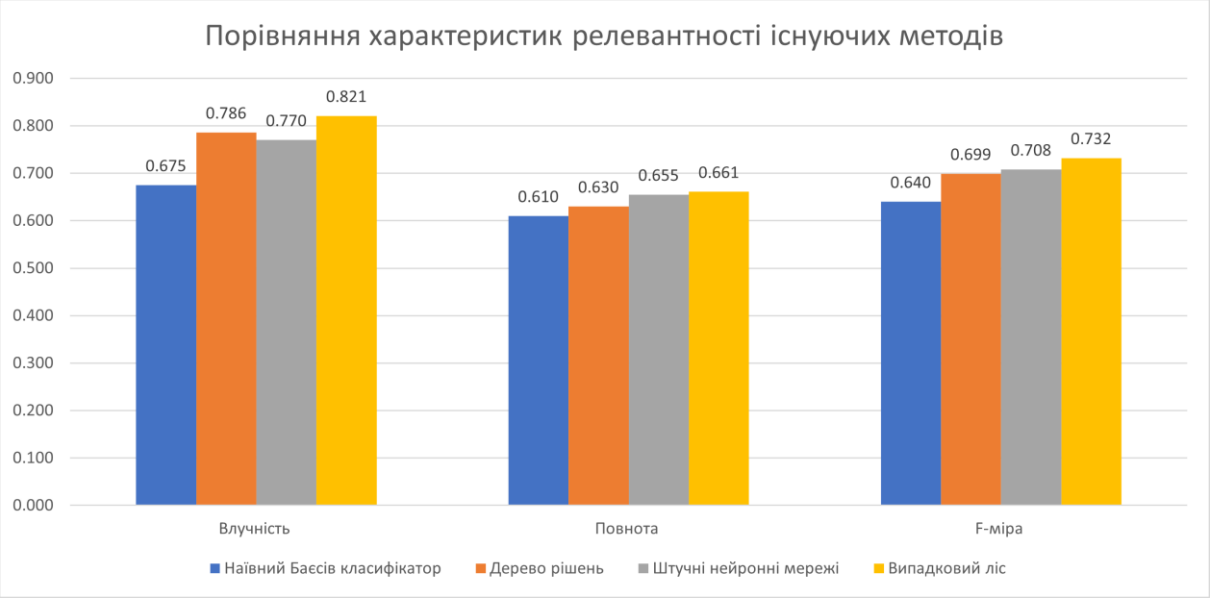
45. A. Panda, S. M. Satapathy, and S. K. Rath, “Empirical Validation of Neural Network Models for Agile Software Effort Estimation based on Story Points,” Procedia Computer Science, vol. 57, pp. 772–781, 2015.

ДОДАТКИ

Додаток 1

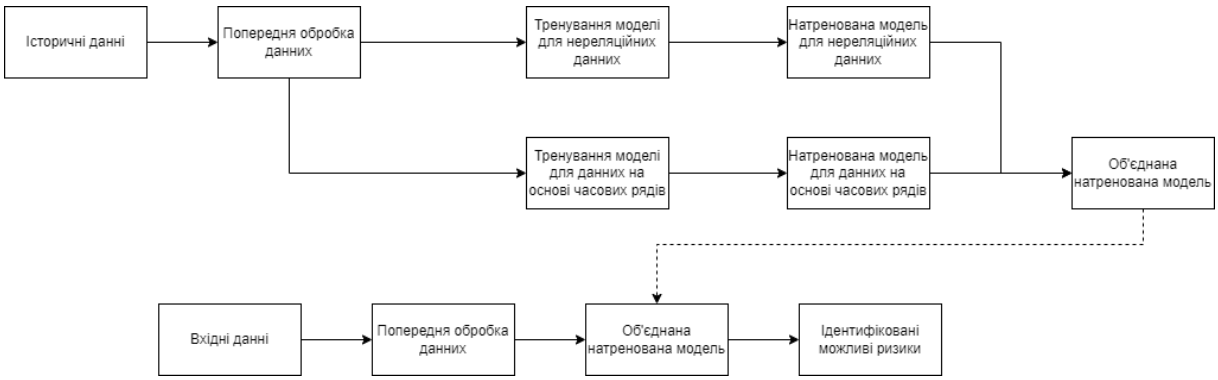
Копії графічних матеріалів

Графік порівняння характеристик релевантності існуючих методів



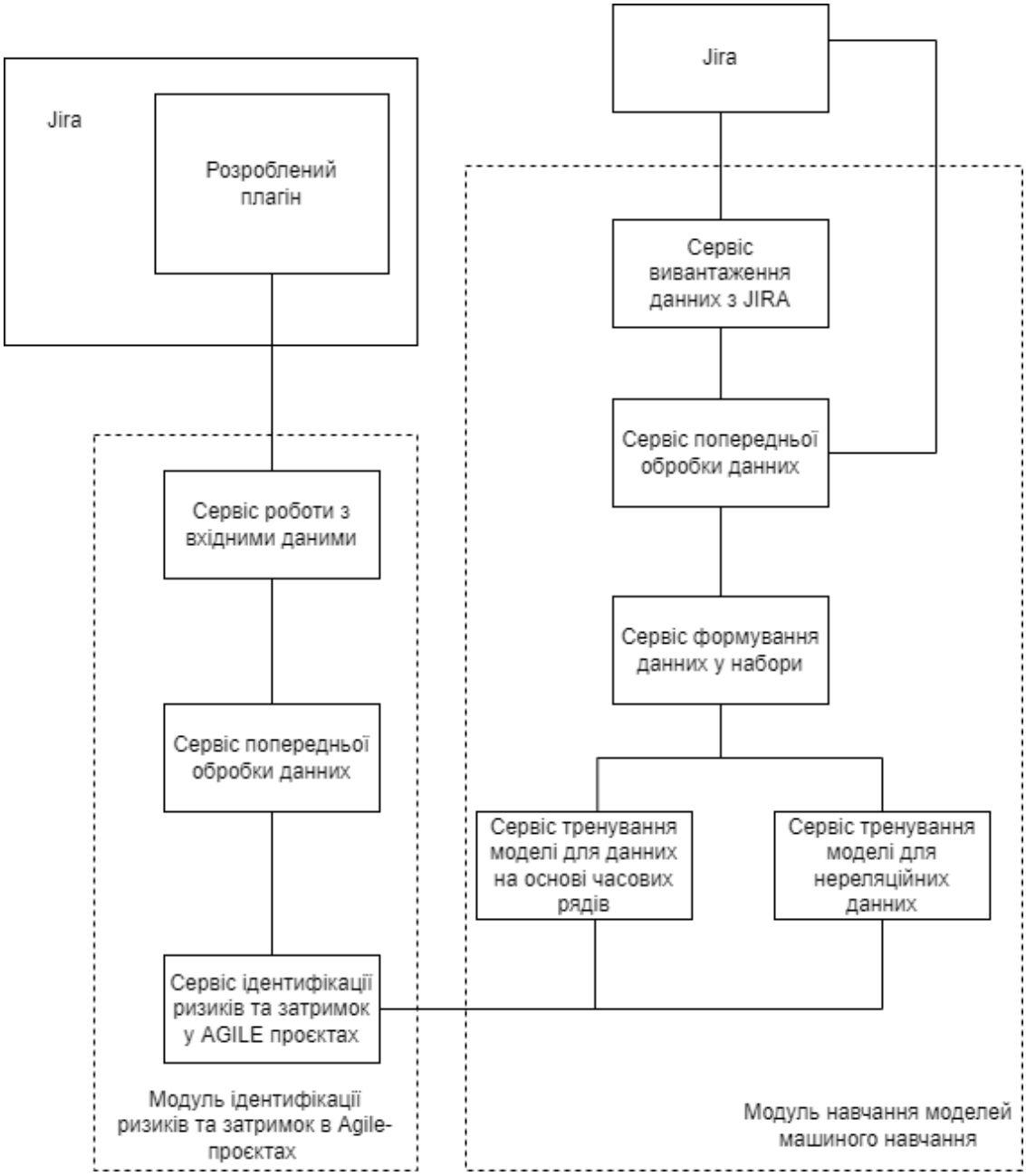
Тютюнник П.Б., КП-11мп

Схема потоку даних запропонованого методу



Тютюнник П.Б., КП-11мп

Схема модулів розробленого програмного забезпечення

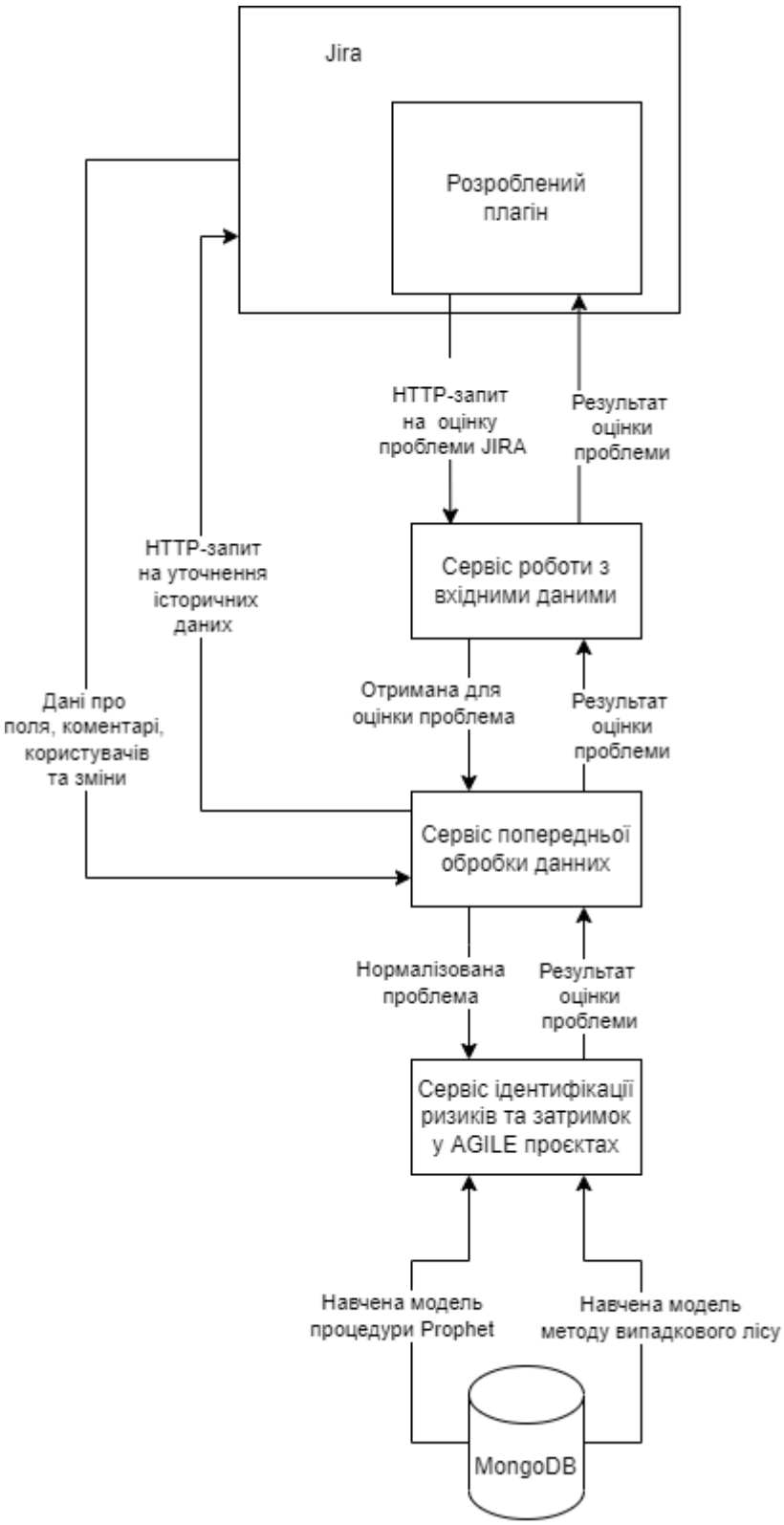


Тютюнник П.Б., КП-11мп

Схема роботи модуля навчання моделей машинного навчання

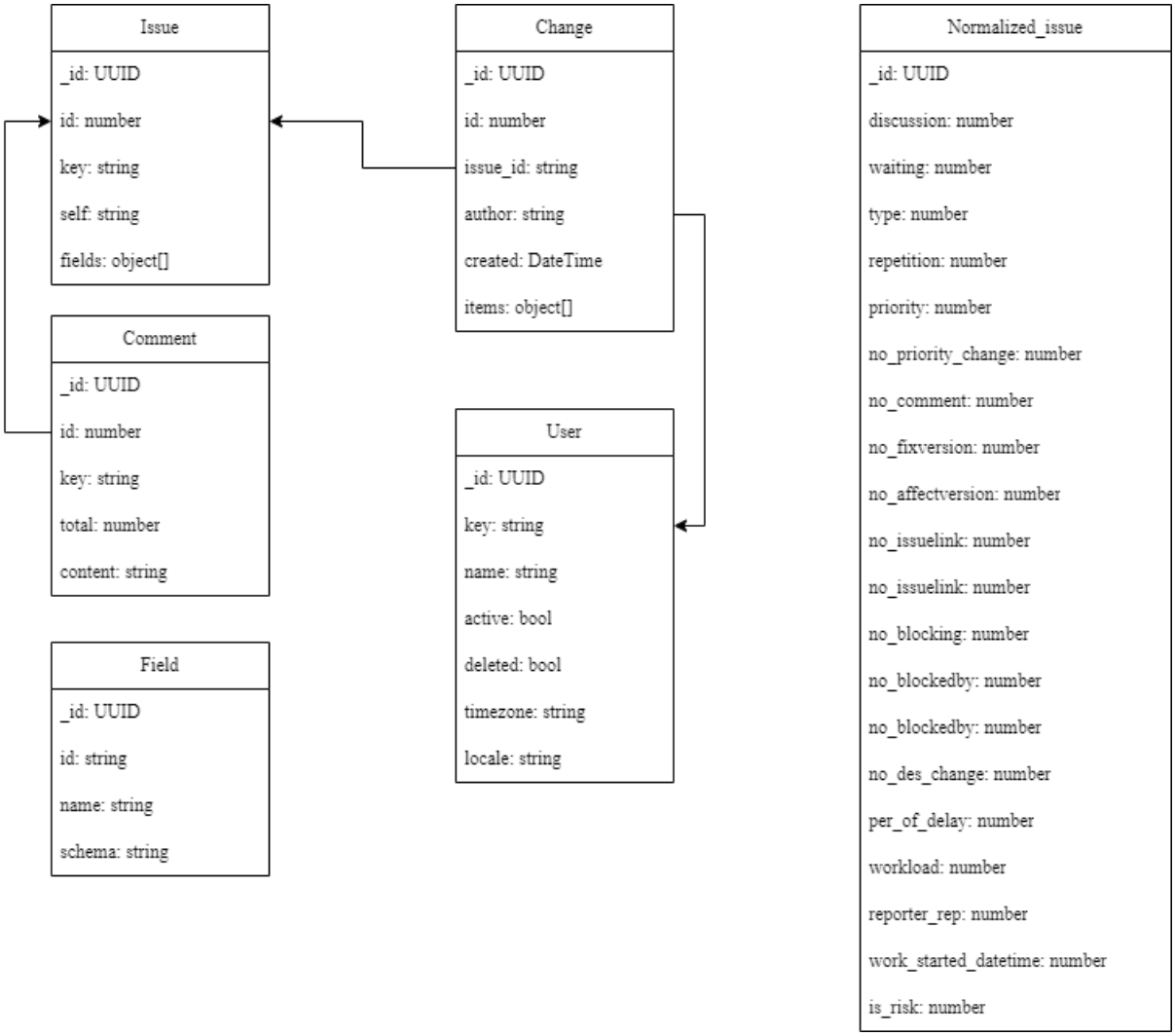
Тютюнник П.Б., КП-11мп

Схема роботи модуля ідентифікації ризиків і затримок у Agile-проєктах



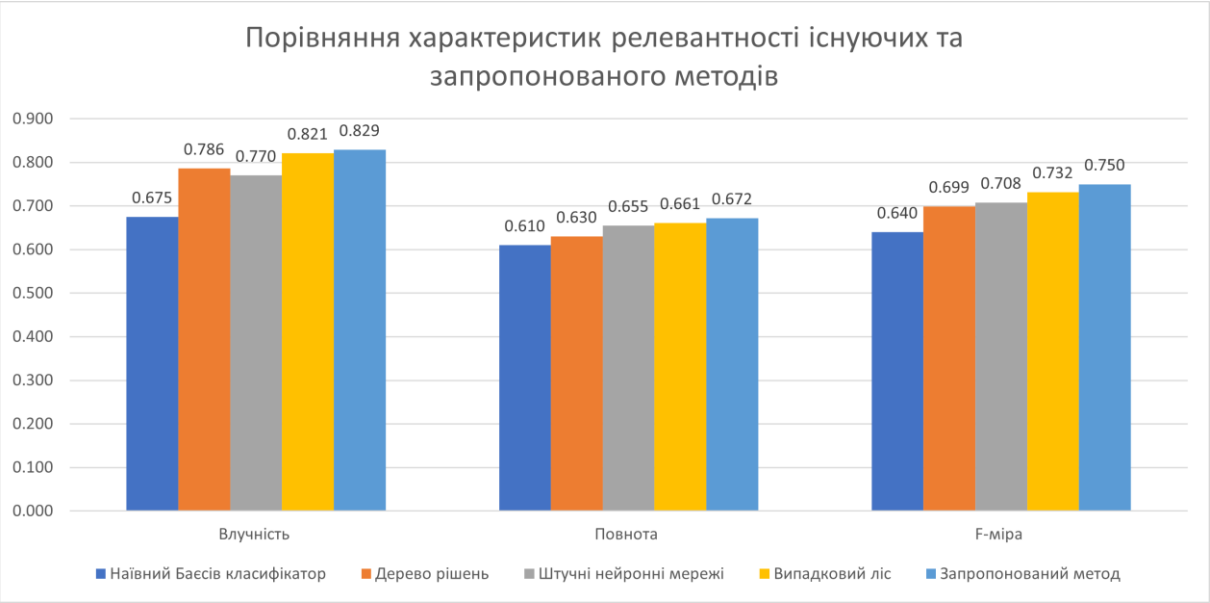
Тютюнник П.Б., КП-11мп

ERD-діаграма сутностей нереляційної БД MongoDB



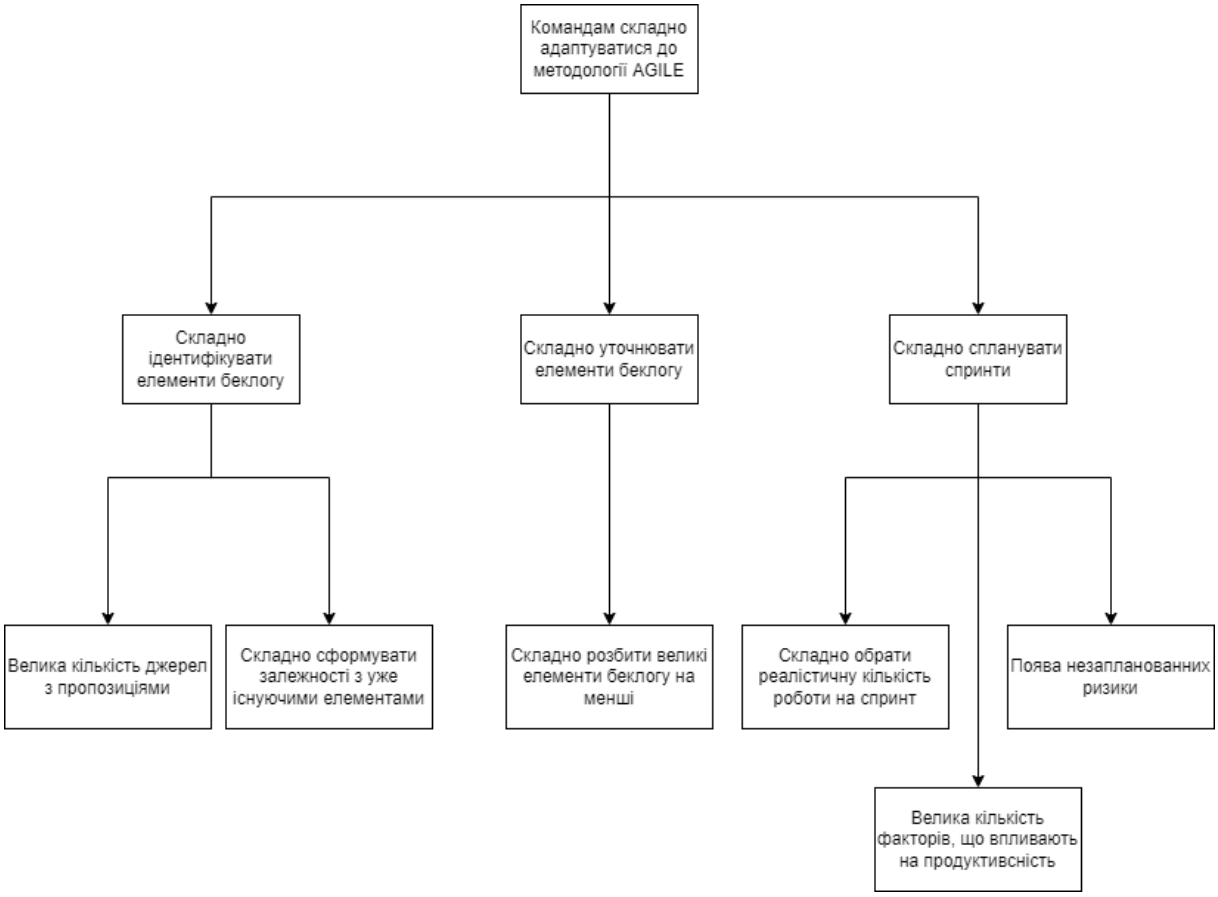
Тютюнник П.Б., КП-11мп

Графік порівняння характеристик релевантності існуючих та запропонованого методів



Тютюнник П.Б., КП-11мп

Дерево проблем



Тютюнник П.Б., КП-11мп

Додаток 2

Лістинг програми

```
const { MongoClient } = require("mongodb");

const axios = require('axios');

axios.get(`https://issues.apache.org/jira/rest/api/2/issue/13471025/comment
`).then(comment => console.log(comment.data));

const uri =
  "mongodb://AzureDiamond:hunter2@localhost:27017/?authSource=admin&readPre
ference=primary&directConnection=true&ssl=false";
const client = new MongoClient(uri);

async function save(project, issue) {
    const database = client.db(project.name);
    const collection = database.collection('comments');

    try {
        const response = await axios.get(`${issue.self}/comment`);

        const data = response.data;

        data.id = issue.id;
        data.key = issue.key;

        await collection.insertOne(data);
    } catch (e) {
        console.log(e);
        await sleep(2000);
    }
}

function sleep(ms) {
    return new Promise((resolve) => {
      setTimeout(resolve, ms);
    });
  }

async function run(project){

   console.log(`Total in ${project.name}: ${total}`);

   let current = project.startAt;

   const database = client.db(project.name);

   const collection = database.collection('issues_beatiful');

   const issues = await collection.find();

   let i = 0;

   console.log(project.name);

   const docs = [];

   for await (const doc of issues) {
        i++;

        if (i < project.startAt){
            continue;
        }

        docs.push(doc);
```

```
        if (docs.length >= 25) {
            const promises = docs.map(async d => await save(project, d));

            await Promise.all(promises);

            docs.splice(0, docs.length);

            console.info(`${project.name}: ${i}`);

            await sleep(500);
        }
    }
}

const issueProviders = [
    {
        name: 'apache',
        url: 'https://issues.apache.org/jira/rest/api/2/issue',
        startAt: 53665
    },
    {
        name: 'moodle',
        url: 'https://tracker.moodle.org/rest/api/2/issue',
        startAt: 50691
    },
    {
        name: 'red_hat',
        url: 'https://issues.redhat.com/rest/api/2/issue',
        startAt: 52225
    },
    {
        name: 'spring',
        url: 'https://jira.spring.io/rest/api/2/issue',
        startAt: 39681
    },
];

const issues = issueProviders.map(element =>
    run(element).catch(error => {
            console.error(error);
            })
);

Promise.all(issues).then(() => console.log('finished'));


// Replace the uri string with your connection string.

axios
  .get('https://issues.apache.org/jira/rest/api/2/search')
  .then(async res => {
    console.log(`statusCode: ${res.status}`);
    //console.log(res.data);

    await save(res.data.issues);
  })
  .catch(error => {
    console.error(error);
  });

const { MongoClient } = require("mongodb");

const fs = require('fs/promises');
```

```
const uri =
  "mongodb://AzureDiamond:hunter2@localhost:27017/?authSource=admin&readPre
ference=primary&directConnection=true&ssl=false";
const client = new MongoClient(uri);

async function collect(project) {
    const obj = {};

    const database = client.db(project.name);
    const collection = database.collection('issues');

    const issues = await collection.find();

    let i = 0;

    console.log(project.name);

    for await (const doc of issues) {
        for (const key in doc.fields) {
            if (Object.hasOwnProperty.call(doc.fields, key)) {
                const element = doc.fields[key];

                if (obj[key] === undefined || obj[key] === null){
                    obj[key] = element;
                }
            }
        }

        if (i % 2000 === 0){
            console.info(i);
        }

        i++;
    }

    console.log(obj);

    var jsonStr = JSON.stringify(obj, null, 4);

    fs.opendir()

    await fs.mkdir('fields');

    await fs.writeFile(`fields/${project.name}.json`, jsonStr);

    for (const key in found) {
        if (Object.hasOwnProperty.call(found, key)) {
            const element = found[key];

            console.log(`${key}: ${element}`);
        }
    }

    console.log(await collection.stats());
}

const issueProviders = [
    {
        name: 'moodle',
        url: 'https://tracker.moodle.org/rest/api/2/search'
    },
    {
        name: 'red_hat',
        url: 'https://issues.redhat.com/rest/api/2/search'
```

```
    },
    {
        name: 'spring',
        url: 'https://jira.spring.io/rest/api/2/search'
    },
    {
        name: 'apache',
        url: 'https://issues.apache.org/jira/rest/api/2/search'
    },
];

async function run() {
    for (const iterator of issueProviders) {
        await collect(iterator);
    }
}

run()
    .catch(error => {
        console.error(error);
    })
    .finally(async () => await client.close());

const { MongoClient } = require("mongodb");

const axios = require('axios');

const uri =
  "mongodb://AzureDiamond:hunter2@localhost:27017/?authSource=admin&readPre
ference=primary&directConnection=true&ssl=false";
const client = new MongoClient(uri);
async function save(project, issues) {
    if (issues.length == 0) {
        return;
    }

    const database = client.db(project.name);
    const collection = database.collection('issues');

    await collection.insertMany(issues);
}

function sleep(ms) {
    return new Promise((resolve) => {
      setTimeout(resolve, ms);
    });
  }

async function run(project) {
    // const fields = (await axios.get(`${project.url}`)).data;

    const database = client.db(project.name);

    const newCol = database.collection('issues_beatiful_with_due_date');

    // await col.insertMany(fields);
    // const collection = database.collection('issues');

    const beatifulCollection = database.collection('issues_beatiful');

    const issues = await beatifulCollection.find();

    let i = 0;
```

```
    console.log(project.name);

    for await (const doc of issues) {


        if (doc.duedate !== null && doc.duedate !== undefined){
            await newCol.insertOne(doc);
        }

        if (i % 20000 === 0){
            console.info(i);
        }

        i++;
    }

}

const issueProviders = [
    {
        name: 'apache',
        url: 'https://issues.apache.org/jira/rest/api/2/field',
        startAt: 577000
    },
    {
        name: 'red_hat',
        url: 'https://issues.redhat.com/rest/api/2/field'
    },
    {
        name: 'moodle',
        url: 'https://tracker.moodle.org/rest/api/2/field',
        startAt: 66000
    },
    {
        name: 'spring',
        url: 'https://jira.spring.io/rest/api/2/field'
    },
];

const issues = issueProviders.map(element =>
    run(element).catch(error => {
            console.error(error);
            })
);

Promise.all(issues).then(() => console.log('finished')).finally(async () =>
await client.close());

const { MongoClient } = require("mongodb");

const axios = require('axios');

const uri =
  "mongodb://AzureDiamond:hunter2@localhost:27017/?authSource=admin&readPre
ference=primary&directConnection=true&ssl=false";
const client = new MongoClient(uri);
async function save(project, issues) {
    if (issues.length == 0) {
        return;
    }

    const database = client.db(project.name);
    const collection = database.collection('issues');
```

```
    await collection.insertMany(issues);
}

function sleep(ms) {
    return new Promise((resolve) => {
      setTimeout(resolve, ms);
    });
  }

async function run(project){
   const total = (await
axios.get(`${project.url}?maxResults=1&startAt=0`)).data.total;

   console.log(`Total in ${project.name}: ${total}`);

   const step = 500;
   let current = project.startAt;

    while (current <= total) {
        const promises = [];

        for (let index = 0; index < 10 && current <= total; index++) {
            const promise = axios
                .get(`${project.url}?maxResults=${step}&startAt=${current}`
)
                .then(async res => {
                    await save(project, res.data.issues);
                });

            current += step;

            promises.push(promise);
        }

        await Promise.all(promises);

        console.debug(`${project.name}: ${current} / ${total}`);

        await sleep(15000);
    }

   console.log(`${project.name}: ${total} downloaded`);
}

const issueProviders = [
    {
        name: 'apache',
        url: 'https://issues.apache.org/jira/rest/api/2/search',
        startAt: 577000
    },
    {
        name: 'red_hat',
        url: 'https://issues.redhat.com/rest/api/2/search'
    },
    {
        name: 'moodle',
        url: 'https://tracker.moodle.org/rest/api/2/search',
        startAt: 66000
    },
    {
        name: 'spring',
        url: 'https://jira.spring.io/rest/api/2/search'
    },
];
```

```
const issues = issueProviders.map(element =>
    run(element).catch(error => {
            console.error(error);
            })
);

Promise.all(issues).then(() => console.log('finished'));

Promise.all(issueProviders);

const { MongoClient } = require("mongodb");

const fs = require("fs/promises");

const uri =
  "mongodb://AzureDiamond:hunter2@localhost:27017/?authSource=admin&readPre
ference=primary&directConnection=true&ssl=false";
const client = new MongoClient(uri);

async function getFields(project) {

    const database = client.db(project.name);
    const collection = database.collection('fields');

    const res = [];

    const fields = await collection.find();

    for await (const field of fields) {
        res.push(field);
    }

    return res;
}

const issueProviders = [
    {
        name: 'moodle',
        url: 'https://tracker.moodle.org/rest/api/2/search'
    },
    {
        name: 'red_hat',
        url: 'https://issues.redhat.com/rest/api/2/search'
    },
    {
        name: 'spring',
        url: 'https://jira.spring.io/rest/api/2/search'
    },
    {
        name: 'apache',
        url: 'https://issues.apache.org/jira/rest/api/2/search'
    },
];

async function run() {
    const res = [];

    for (const iterator of issueProviders) {
        const fields = await getFields(iterator);

        res.push({project: iterator.name, fields: fields});
    }
```

```
    const stats = [];

    for (const proj of res) {
        for (const field of proj.fields) {
            if (field.name != undefined){

                if (stats[field.name] != undefined){

                    if (!stats[field.name].find(x => x.project ===
proj.project)){
                        stats[field.name].push({
                            project: proj.project,
                            custom: field.custom,
                            id: field.id,
                            name: field.name
                        });
                    }

                } else {
                    stats[field.name] = [{
                        project: proj.project,
                        custom: field.custom,
                        id: field.id,
                        name: field.name
                    }];
                }
                // console.log(stats[field.name]);
            }

            // console.log(field.name);
        }
    }

    return stats;
}

run()
.then( async res => {


    for (let index = 1; index <= 4; index++) {
        const stat = [];

        for (const key in res) {
            if (Object.hasOwnProperty.call(res, key)) {
                const element = res[key];

                // console.log(element);

                if (element.length === index && index !== 4){
                    stat.push({custom: element[0].custom, id:
element[0].id, name: element[0].name, projects: element.map(el =>
el.project)});
                } else if (element.length === index) {
                    stat.push({custom: element[0].custom, id:
element[0].id, name: element[0].name});
                }
            }
        }

        const str = JSON.stringify(stat, null, 4);
        await fs.writeFile(`fields_${index}.json`, str);
    }
} )
```

```
    .catch(error => {
        console.error(error);
    })
    .finally(async () => await client.close());

const { MongoClient } = require("mongodb");

const fs = require("fs/promises");

const uri =
  "mongodb://AzureDiamond:hunter2@localhost:27017/?authSource=admin&readPre
ference=primary&directConnection=true&ssl=false";
const client = new MongoClient(uri);

async function getFields(project) {

    const database = client.db(project.name);
    const collection = database.collection('fields');

    const res = [];

    const fields = await collection.find();

    for await (const field of fields) {
        res.push(field);
    }

    return res;
}

const issueProviders = [
    {
        name: 'moodle',
        url: 'https://tracker.moodle.org/rest/api/2/search'
    },
    {
        name: 'red_hat',
        url: 'https://issues.redhat.com/rest/api/2/search'
    },
    {
        name: 'spring',
        url: 'https://jira.spring.io/rest/api/2/search'
    },
    {
        name: 'apache',
        url: 'https://issues.apache.org/jira/rest/api/2/search'
    },
];

async function run() {
    const res = [];

    for (const iterator of issueProviders) {
        const fields = await getFields(iterator);

        res.push({project: iterator.name, fields: fields});
    }

    const stats = [];

    for (const proj of res) {
        for (const field of proj.fields) {
            if (field.name != undefined){
```

```
                if (stats[field.name] != undefined){

                    if (!stats[field.name].find(x => x.project ===
proj.project)){
                        stats[field.name].push({
                            project: proj.project,
                            custom: field.custom,
                            id: field.id,
                            name: field.name
                        });
                    }

                } else {
                    stats[field.name] = [{
                        project: proj.project,
                        custom: field.custom,
                        id: field.id,
                        name: field.name
                    }];
                }
                // console.log(stats[field.name]);
            }

            // console.log(field.name);
        }
    }

    return stats;
}

run()
.then( async res => {


    for (let index = 1; index <= 4; index++) {
        const stat = [];

        for (const key in res) {
            if (Object.hasOwnProperty.call(res, key)) {
                const element = res[key];

                // console.log(element);

                if (element.length === index && index !== 4){
                    stat.push({custom: element[0].custom, id:
element[0].id, name: element[0].name, projects: element.map(el =>
el.project)});
                } else if (element.length === index) {
                    stat.push({custom: element[0].custom, id:
element[0].id, name: element[0].name});
                }
            }
        }

        const str = JSON.stringify(stat, null, 4);
        await fs.writeFile(`fields_${index}.json`, str);
    }
} )
    .catch(error => {
        console.error(error);
    })
    .finally(async () => await client.close());

version: '3'
```

```
services:
  mongodb:
    image: mongo:5.0
    environment:
      - MONGO_INITDB_ROOT_USERNAME=AzureDiamond
      - MONGO_INITDB_ROOT_PASSWORD=hunter2
    ports:
      - 27017:27017
    volumes:
      - mongodb:/data/db
      - mongodb_config:/data/configdb
volumes:
  mongodb:
  mongodb_config:
```

```
package com.example.plugins.tutorial.jira.customfields;

import com.atlassian.jira.issue.customfields.impl.AbstractSingleFieldType;
import
com.atlassian.jira.issue.customfields.persistence.PersistenceFieldType;
import com.atlassian.plugin.spring.scanner.annotation.component.Scanned;
import com.atlassian.plugin.spring.scanner.annotation.imports.JiraImport;
import com.atlassian.jira.issue.customfields.manager.GenericConfigManager;
import
com.atlassian.jira.issue.customfields.persistence.CustomFieldValuePersister
;
import com.atlassian.jira.issue.customfields.impl.FieldValidationException;
import java.math.BigDecimal;

@Scanned
public class MoneyCustomField extends AbstractSingleFieldType<BigDecimal> {

    public MoneyCustomField(
            @JiraImport CustomFieldValuePersister
customFieldValuePersister,
            @JiraImport GenericConfigManager genericConfigManager) {

        super(customFieldValuePersister, genericConfigManager);
    }

    @Override
    public String getStringFromSingularObject(final BigDecimal
singularObject) {
        if (singularObject == null)
            return null;
        else
            return singularObject.toString();
    }

    @Override
    public BigDecimal getSingularObjectFromString(final String string)
throws FieldValidationException {
        if (string == null)
            return null;
        try {
            BigDecimal decimal = new BigDecimal(string);
            if (decimal.scale() > 2) {
                throw new FieldValidationException(
                        "Maximum of 2 decimal places are allowed.");
            }
            return decimal.setScale(2);
        } catch (NumberFormatException ex) {
            throw new FieldValidationException("Not a valid number.");
        }
```

```
    }

    @Override
    protected PersistenceFieldType getDatabaseType() {
        return PersistenceFieldType.TYPE_LIMITED_TEXT;
    }

    @Override
    protected BigDecimal getObjectFromDbValue(final Object databaseValue)
            throws FieldValidationException {
        return getSingularObjectFromString((String) databaseValue);
    }

    @Override
    protected Object getDbValueFromObject(final BigDecimal
customFieldObject) {
        return getStringFromSingularObject(customFieldObject);
    }
}

#if ($value)
    $$value
#end

package com.example.plugins.tutorial.jira.customfields;

import com.atlassian.jira.issue.customfields.impl.FieldValidationException;
import org.junit.Test;

import java.math.BigDecimal;

import static junit.framework.TestCase.assertEquals;
import static junit.framework.TestCase.fail;

public class MoneyCustomFieldTest {

    @Test
    public void testGetDbValueFromObject() throws Exception
    {
        MoneyCustomField moneyCustomField = new MoneyCustomField(null,
null);
        assertEquals("1", moneyCustomField.getDbValueFromObject(new
BigDecimal(1)));
        assertEquals("1.02", moneyCustomField.getDbValueFromObject(new
BigDecimal("1.02")));
        assertEquals("0.02", moneyCustomField.getDbValueFromObject(new
BigDecimal("0.02")));
    }

    @Test
    public void testGetStringFromSingularObject() throws Exception
    {
        MoneyCustomField moneyCustomField = new MoneyCustomField(null,
null);
        assertEquals("1", moneyCustomField.getStringFromSingularObject(new
BigDecimal(1)));
        assertEquals("1.02",
moneyCustomField.getStringFromSingularObject(new BigDecimal("1.02")));
        assertEquals("0.02",
moneyCustomField.getStringFromSingularObject(new BigDecimal("0.02")));
    }

    @Test
    public void testGetSingularObjectFromString() throws Exception
```

```
    {
        MoneyCustomField moneyCustomField = new MoneyCustomField(null,
null);
        assertEquals("3.00",
moneyCustomField.getSingularObjectFromString("3").toString());
        assertEquals("3.03",
moneyCustomField.getSingularObjectFromString("3.03").toString());
        assertEquals("3.20",
moneyCustomField.getSingularObjectFromString("3.2").toString());
        // Now test the errors:
        try
        {
            moneyCustomField.getSingularObjectFromString("3 dollars");
            fail("Validation should have failed.");
        }
        catch (FieldValidationException ex)
        {
            assertEquals("Not a valid number.", ex.getMessage());
        }
        try
        {
            moneyCustomField.getSingularObjectFromString("3.203");
            fail("Validation should have failed.");
        }
        catch (FieldValidationException ex)
        {
            assertEquals("Maximum of 2 decimal places are allowed.",
ex.getMessage());
        }
    }

}
```

```
#customControlHeader ($action $customField.id $customField.name
$fieldLayoutItem.required $displayParameters $auiparams)
<input class="text" id="$customField.id" name="$customField.id" type="text"
value="$textutils.htmlEncode($!value)" />
#customControlFooter ($action $customField.id
$fieldLayoutItem.fieldDescription $displayParameters $auiparams)
from fastapi import APIRouter, Body, Request, Response, HTTPException,
status
from fastapi.encoders import jsonable_encoder
from typing import List

from models import Book, BookUpdate

router = APIRouter()

from sklearn.ensemble import RandomForestClassifier
X = list(router.app.database["training"].find())
Y = list(router.app.database["testing"].find())

clf = RandomForestClassifier(n_estimators=100)
clf = clf.fit(X, Y)

import pandas as pd
from prophet import Prophet

prophetSet = router.app.database["training_csv"].find(1)

df = pd.read_csv(prophetSet.training_csv_data)
```

```
m = Prophet()
m.fit(df)
```

Додаток 3

Копія презентації

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

КАФЕДРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ КОМП’ЮТЕРНИХ СИСТЕМ

ПРОГРАМНИЙ МЕТОД ІДЕНТИФІКАЦІЇ РИЗИКІВ ТА ЗАТРИМОК У AGILE-ПРОЄКТАХ

Виконав: Тютюнник Петро Богданович

Науковий керівник: доцент, к.т.н., Рибачок Наталія Антонівна

Київ – 2022

АКТУАЛЬНІСТЬ ДОСЛІДЖЕННЯ

- методологією Agile користуються хоча б частково до 95% технологічних компаній;
- з переходом на віддалену роботу кількість команд, що працюють за методологією виросла більше ніж у 2 рази (з 37 до 86% серед розробників ПЗ);
- основною названою причиною невдалих Agile-проєктів є ризики, які не були виявлені на етапі планування, і великі затримки;
- існуючі методи виявлення ризиків і затримок не є автоматизованими і, отже, більш схильні до людських помилок. Часто проєктні команди не можуть вчасно визначити необхідні ризики, що призводить до провалу проєктів.

Об’єкт дослідження: процес ідентифікації ризиків та затримок у Agile-проєктах.

Предмет дослідження: методи та програмні засоби ідентифікації ризиків та затримок у AGILE проєктах.

Мета дослідження: покращити показники влучності, повноти та F-міри для задачі ідентифікації ризиків та затримок у Agile-проєктах, шляхом розробки відповідних методу та ПЗ.

Окремі завдання

1. Дослідити існуючі програмні методи ідентифікації ризиків та затримок у Agile-проєктах.
2. Розробити модифікований метод ідентифікації ризиків та затримок у Agile-проєктах.
3. Розробити програмне забезпечення, що реалізує запропонований метод.
4. Зібрати інформацію про проблеми Agile-проєктів з відкритим програмним кодом та сформувати набори даних.
5. Проаналізувати характеристики релевантності запропонованого методу у порівнянні з існуючими методами.

Ризик

Ризик – невизначена подія або умова, які, якщо вони відбуваються, мають позитивний або негативний вплив на цілі проєкту. Серед негативних впливів найбільш поширеними є затримка в часі виконання проєкту або необхідність в додаткових ресурсах.

$$Risk = Likelihood \cdot Impact$$

Overall Risk Severity				
Impact	HIGH	Medium	High	Critical
	MEDIUM	Low	Medium	High
	LOW	None	Low	Medium
		LOW	MEDIUM	HIGH
	Likelihood			

НАЙБІЛЬШ ВІДОМІ РІШЕННЯ ЦІЄЇ НАУКОВОЇ ПРОБЛЕМИ

Для вирішення цієї проблеми використовуються методи машинного навчання, оскільки це проблема бінарної класифікації, де необхідно з’ясувати, чи належить проблема AGILE-проєкту до набору можливих ризиків чи ні.

Для вирішення цієї проблеми застосовуються наступні методи:

- випадковий ліс;
- дерево рішень;
- наївний Баєсів класифікатор;
- штучні нейронні мережі.

ПОРІВНЯННЯ ІСНУЮЧИХ РІШЕНЬ

Для визначення найефективнішого методу класифікації ризиків у Agile-проєктах було проведено експеримент, застосовуючи створений з проблем проєктів з відкритим кодом датасет.

Для оцінки ефективності будемо використовувати наступні характеристки релевантності:

- Влучність (precision);
- Повнота (recall);
- F-міра.

ПОРІВНЯННЯ ІСНУЮЧИХ РІШЕНЬ

Характериситки релевантності обчислюємо за наступними формулами:

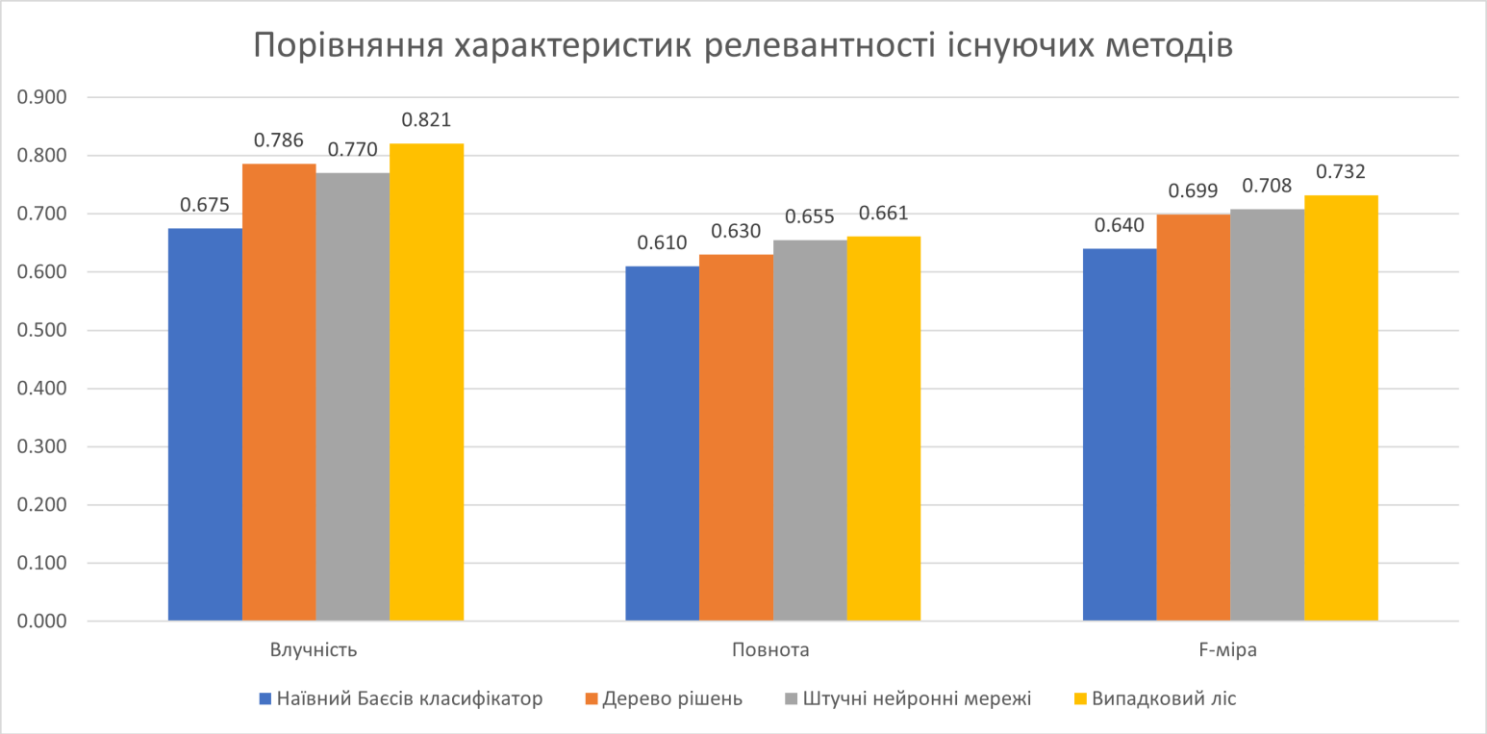
$$P = \frac{t_p}{t_p + f_p},$$

$$R = \frac{t_p}{t_p + f_n},$$

$$F = 2 \cdot \frac{P \cdot R}{P + R},$$

де P – влучність, R – повнота, F – F-міра відповідно, t_p – кількість істинно-позитивних спрацювань, f_p – кількість хибно-позитивних спрацювань, f_n – кількість хибно-негативних спрацювань.

ПОРІВНЯННЯ ІСНУЮЧИХ РІШЕНЬ



Для оцінки використовувалися реалізації методів з бібіліотеки *scikit-learn*. Найвищі характеристики релевантності має метод випадкового лісу.

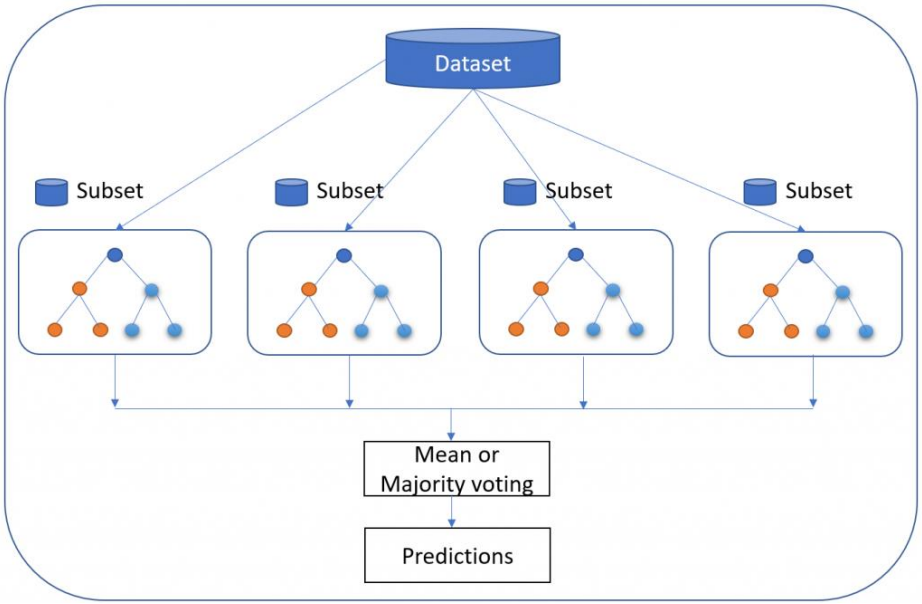
Недоліком усіх розглянутих методів є відсутність можливості оцінки параметрів у форматі дат.

ГІПОТЕЗИ

Поєднання методу *випадкового лісу* та процедури *Prophet* дозволить розробити метод ідентифікації ризиків та затримок у Agile-проєктах з вищими показниками влучності, повноти та F-міри, за рахунок включення даних на основі дат.

Випадковий ліс

Випадковий ліс прогнозує значення відповіді шляхом усереднення прогнозованих значень відповіді по багатьох деревах рішень. Кожне дерево вирощується на початковому зразку навчальних даних. Завантажувальна вибірка – це випадкова вибірка спостережень, зібрана із заміною. Крім того, предиктори відбираються під час кожного розбиття в дереві рішень. Дерево рішень підбирається за допомогою методології рекурсивного розбиття. Отримані дерева рішень потім використовуються для класифікації даних, де результатом випадкового лісу є клас, що обраний більшістю дерев.



Випадковий ліс

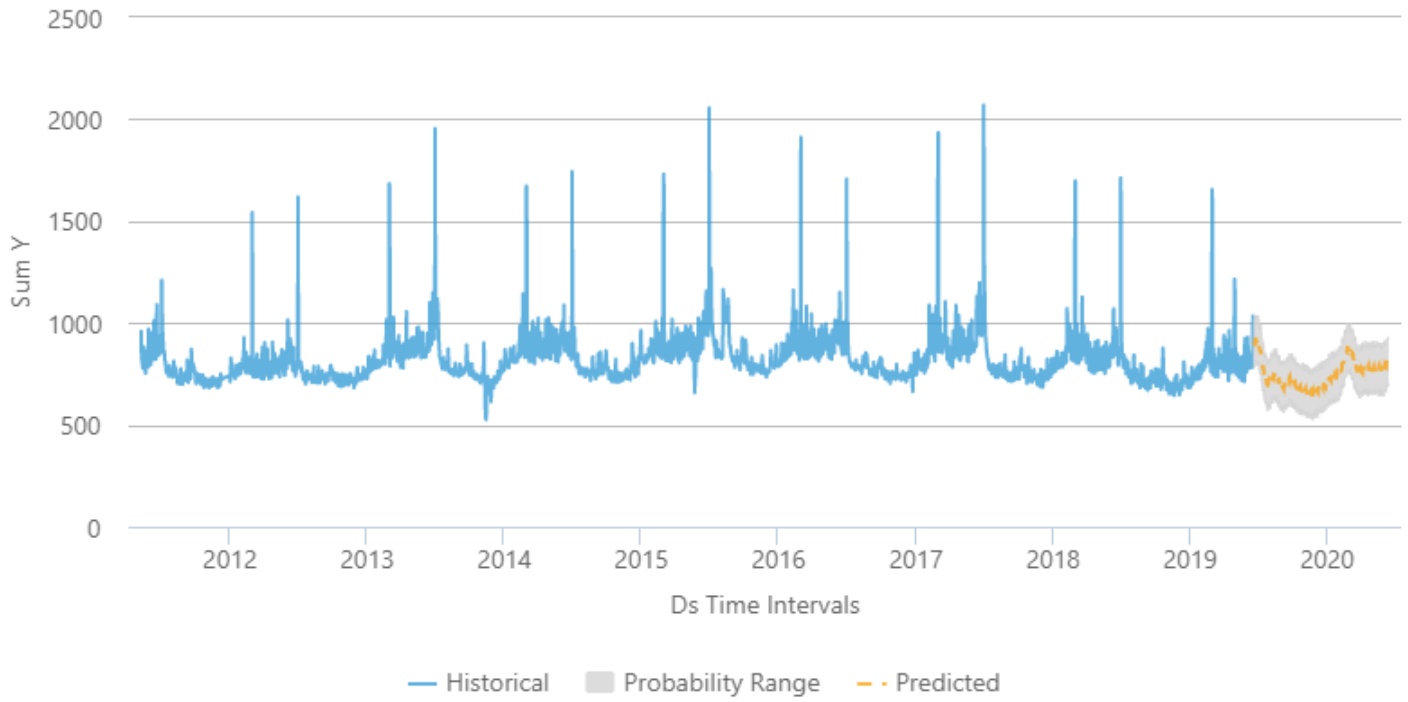
Набори даних використовувалися для навчання моделі методу випадкового лісу. Для визначення результату набору дерев рішень використовувалась наступна функція:

$$P(x) = \frac{\sum_{i=0}^{n} P_i(x)}{n},$$

де $P(x)$ – результат класифікації для проблеми x, $P_i(x)$ – результат класифікації для проблеми x для піддерева i, n – кількість піддерев в ансамблевому методі.

Prophet

Prophet — це процедура прогнозування даних часових рядів на основі адитивної моделі, де нелінійні тенденції відповідають річній, тижневій і щоденній сезонності, а також ефектам свят.



Запропонований метод

Щоб включити дані, пов’язані з датою, модель Prophet була навчена передбачати ймовірність того, яка проблема стане ризикованою, якщо робота над проблемою починається у відповідну дату. Середня ймовірність того, що проблема буде затриманою та становитиме ризик, становить приблизно 16,7%, але для проблем, які почалися наприкінці кварталів або під час святкових сезонів, такий шанс вищий, ніж середній наприклад 17,4% для різдвяних свят. Таким чином, можна припустити, що використання коефіцієнта, який дорівнює частці ділення прогнозованої ймовірності ризику моделі Prophet на середню ймовірність, для зміни результатів розподілу в наборах дерева рішень.

$$k(x) = \frac{k_{prognosed}(x)}{k_{avg}}$$

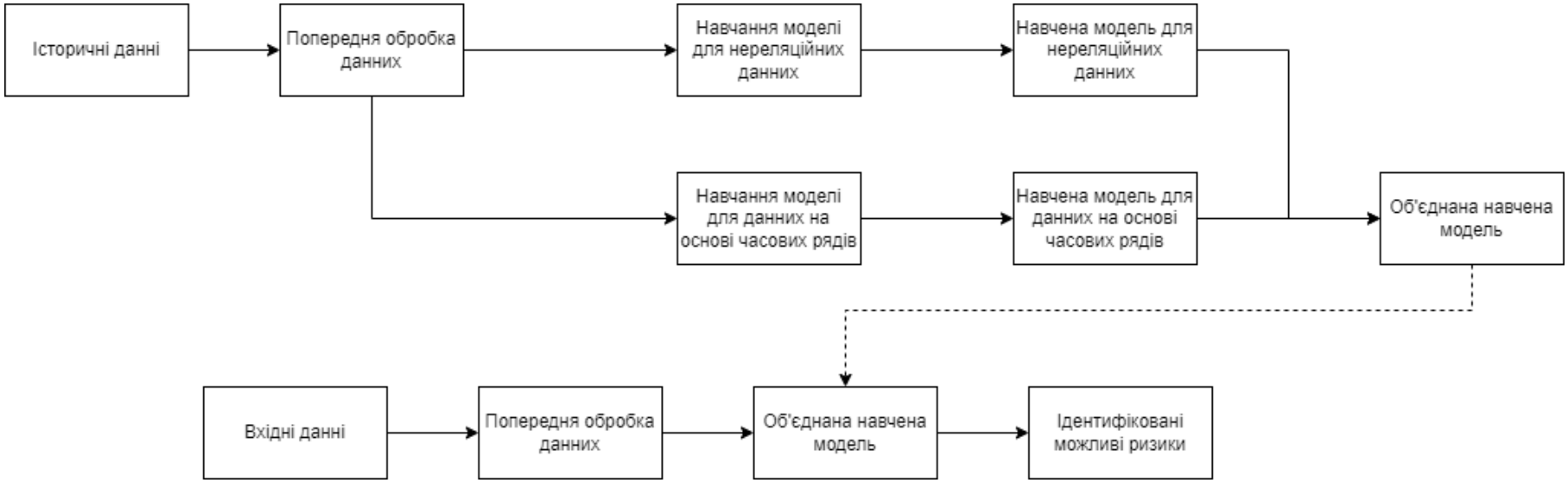
Запропонований метод

Для визначення результату набору дерев рішень для запропонованого методу використовувалась наступна функція:

$$P(x) = \frac{k_{prognosed}(x)}{k_{avg}} \cdot \frac{\sum_{i=0}^{n} P_i(x)}{n},$$

де $P(x)$ – результат класифікації для проблеми x, $P_i(x)$ – результат класифікації для проблеми x для піддерева i, n – кількість піддерев в ансамблевому методі, $k_{prognosed}(x)$ – прогнозована процедурою Prophet ймовірність, що проблема x є ризиком, k_{avg} – середня ймовірність що проблема є можливим ризиком.

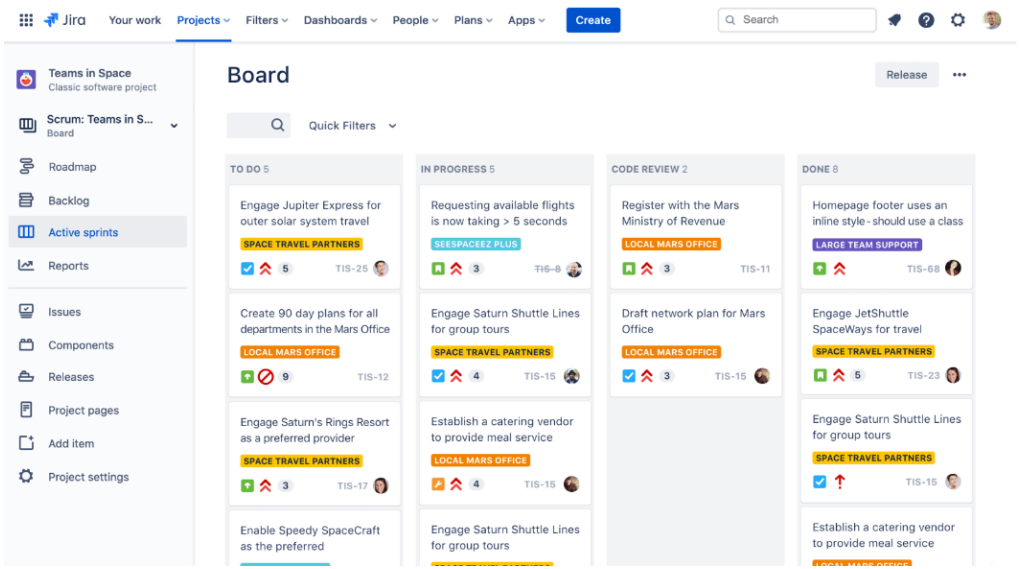
Запропонований метод



Вхідні данні

Використовуються данні з кількох проєктів з відкритим кодом, які використовують Atlassian JIRA для відстеження помилок та завдань.

Atlassian JIRA — система відстеження помилок, призначена для організації спілкування з користувачами, і для управління проєктами. Розроблена компанією Atlassian в 2002 році.



Вхідні данні

Проєкти з відкритим кодом, що використовують Atlassian JIRA:

- Apache;
- Moodle;
- Red Hat;
- Spring.

У сумі для 4 проєктів 1,5+ мільйона проблем, але з яких лише невелику частину з них можна було використати для навчання методу через відсутність необхідних даних для визначення статусу затримки цих проблем.

Отриманий набір даних складається з 50 тисяч попередньо оброблених проблем.

Вхідні данні

Проєкт	Набір для навчання			Набір для тестування		
	Є ризиком	Не є ризиком	Усього	Є ризиком	Не є ризиком	Усього
Apache	1246	5228	6474	461	1615	2076
Moodle	1319	6470	7789	359	1420	1778
Red Hat	3326	16,489	19815	812	3,326	4138
Spring	539	3839	4378	326	1296	1622
Усього:			38456			9614

Отриманий датасет був розбитий на набір для навчання та набір для тестування у співвідношенні 80 до 20 відсотків відповідно, де старіші данні використовуються для навчання, а новіші для тестування.

Параметри

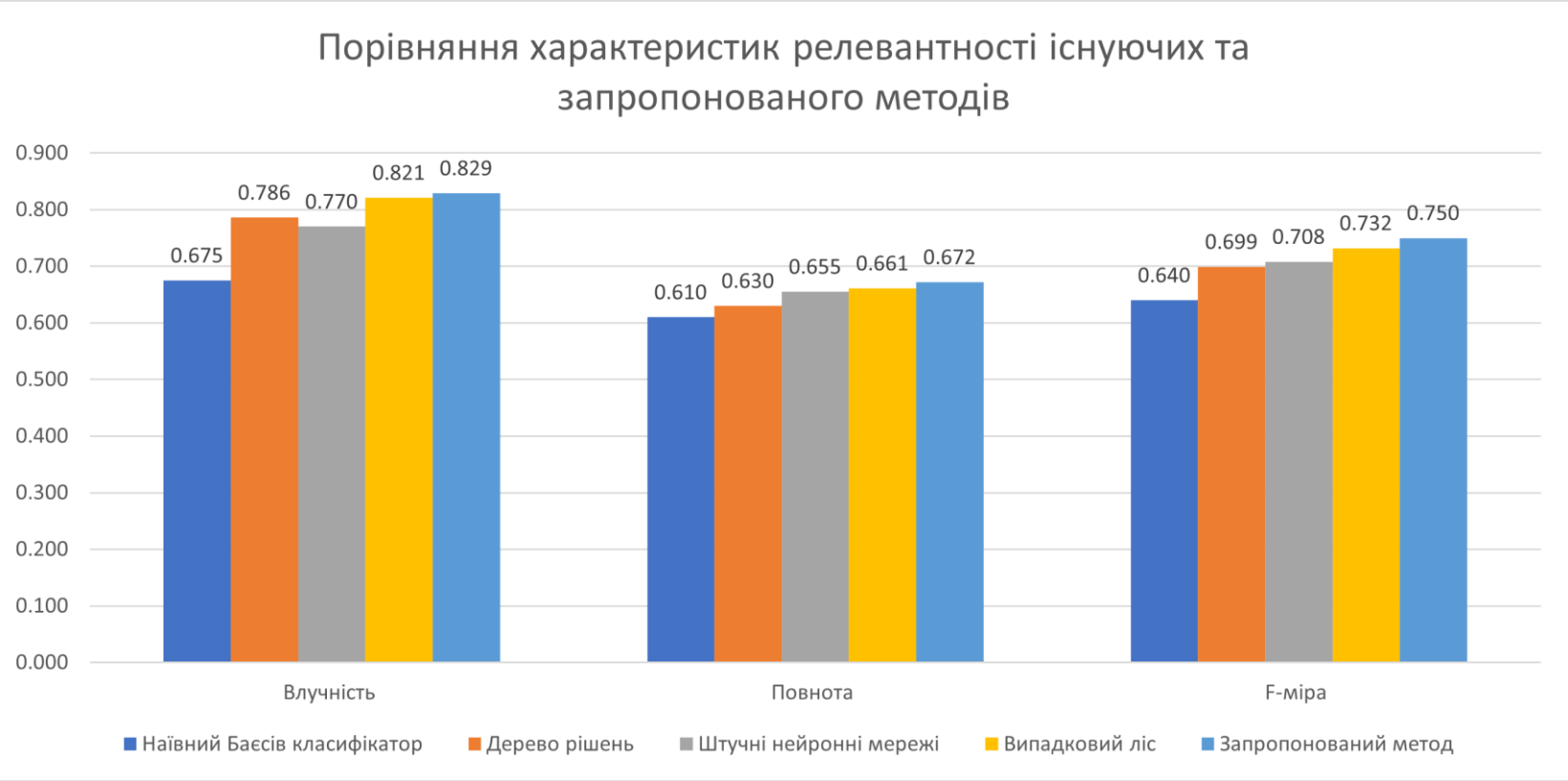
Проблема (issue) залежно від проєкту має різну структуру і для вхідних даних були визначенні наступні параметри, для яких було виконано l_1 логістичну регресію для позначення впливу на факт ризику:

Параметр	Результат регресії
Тип: Баг	–0,904
Тип: Документація	1,16
Тип: Покращення	0,247
Тип: Новий функціонал	0,83
Тип: Історія	–0,528
Тип: Підзавдання	1,637
Тип: Завдання	0,783
Пріоритет: Блокер	–0,748
Пріоритет: Критичний	–0,947
Пріоритет: Вагомий	–0,901
Пріоритет: мінімальний	–0,704
Пріоритет: другорядний	–0,764

Параметри

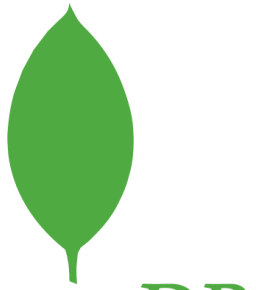
Параметр	Результат регресії
Час обговорення	–6,974
Час очікування	–4,689
Кількість разів зміни пріоритету	1,434
Кількість разів повторного відкриття проблеми	3,016
Кількість коментарів	2,963
Кількість виправлених версій	2,207
Кількість версій з проблемою	–3,201
Кількість пов’язаних проблем	1,415
Кількість проблем, які блокуються цією проблемою	0,216
Кількість проблем, які блокують цю проблему	1,935
Кількість зміни опису	1,678
Репутація репортера	–0,683
Робоче навантаження розробника	1,791
Відсоток проблем із затримкою, якими займався розробник	3,497

Оцінка запропонованого методу

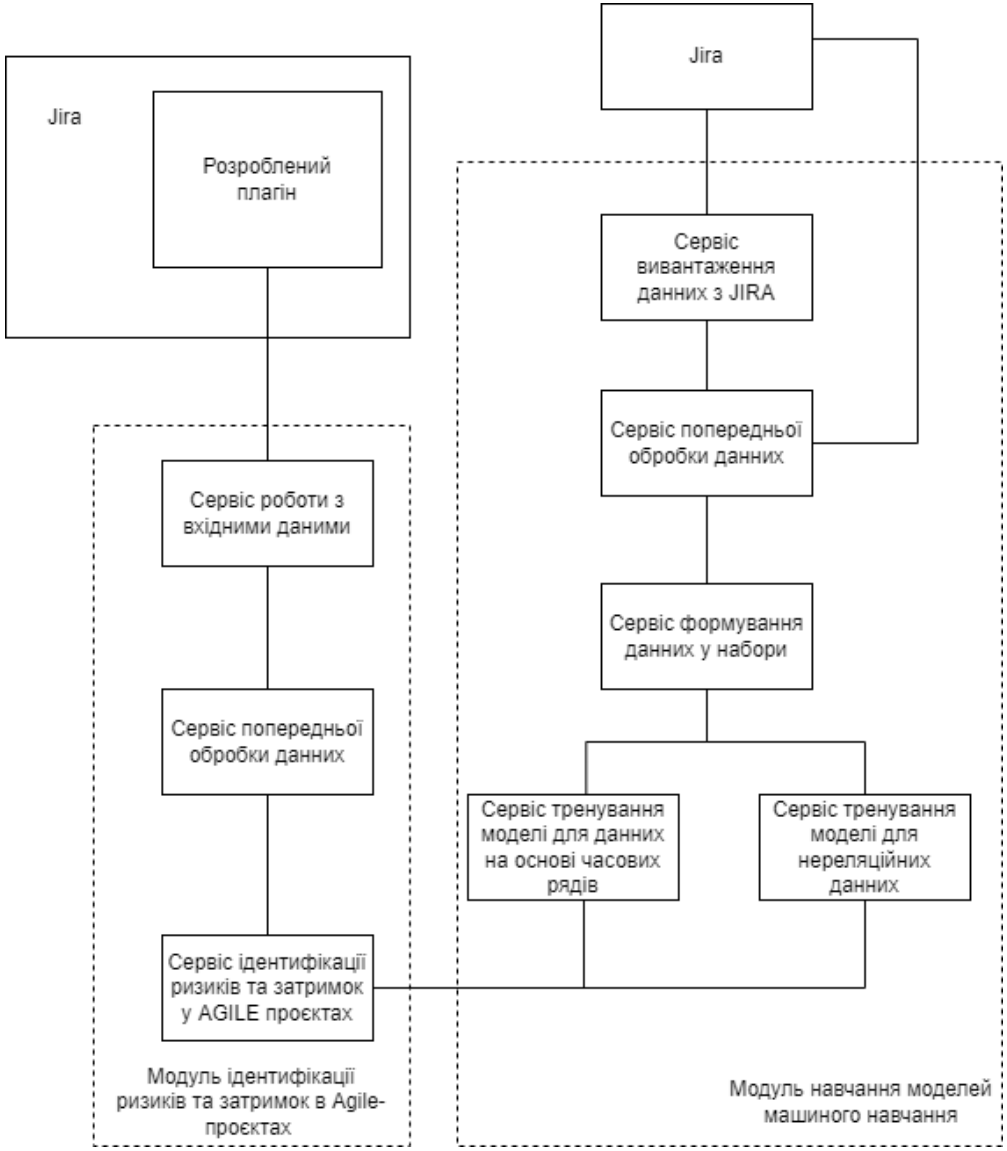


Для оцінки використовувалися реалізації методів з бібліотеки *scikit-learn*
Запропонований метод показує характеристики релевантності на 0.01-0.02 вищі ніж базовий метод.

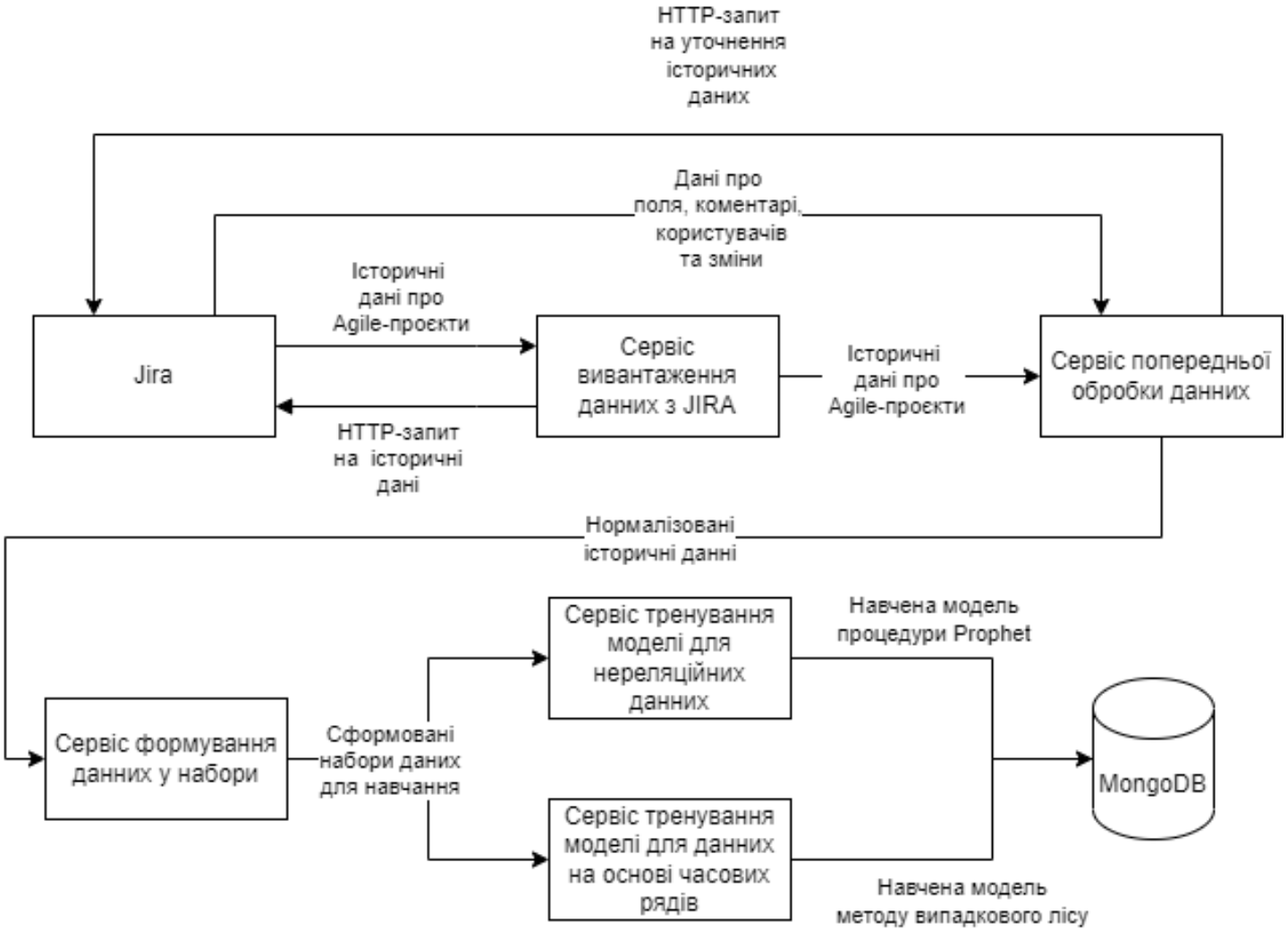
Засоби реалізації



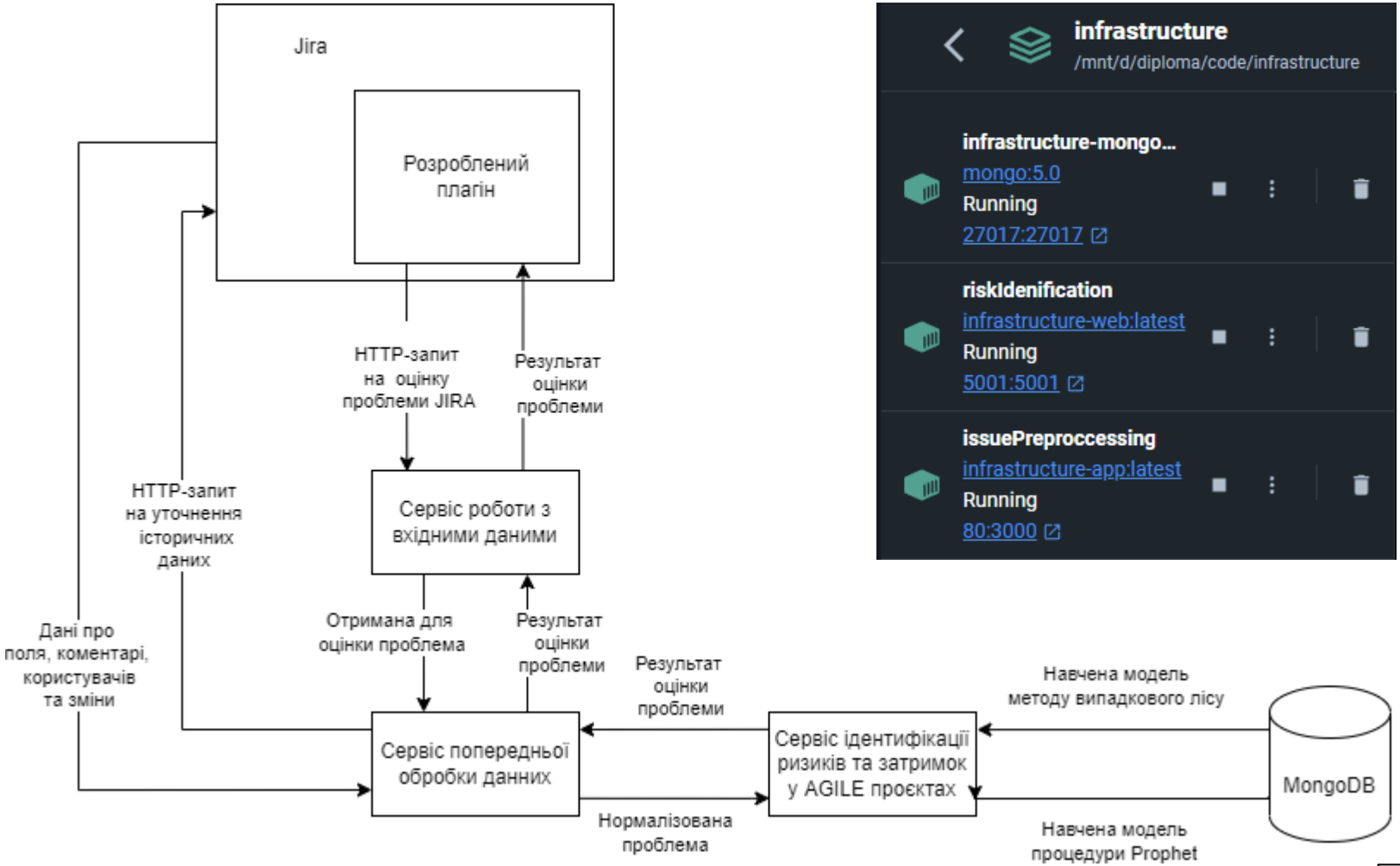
Архітектура програмної реалізації методу



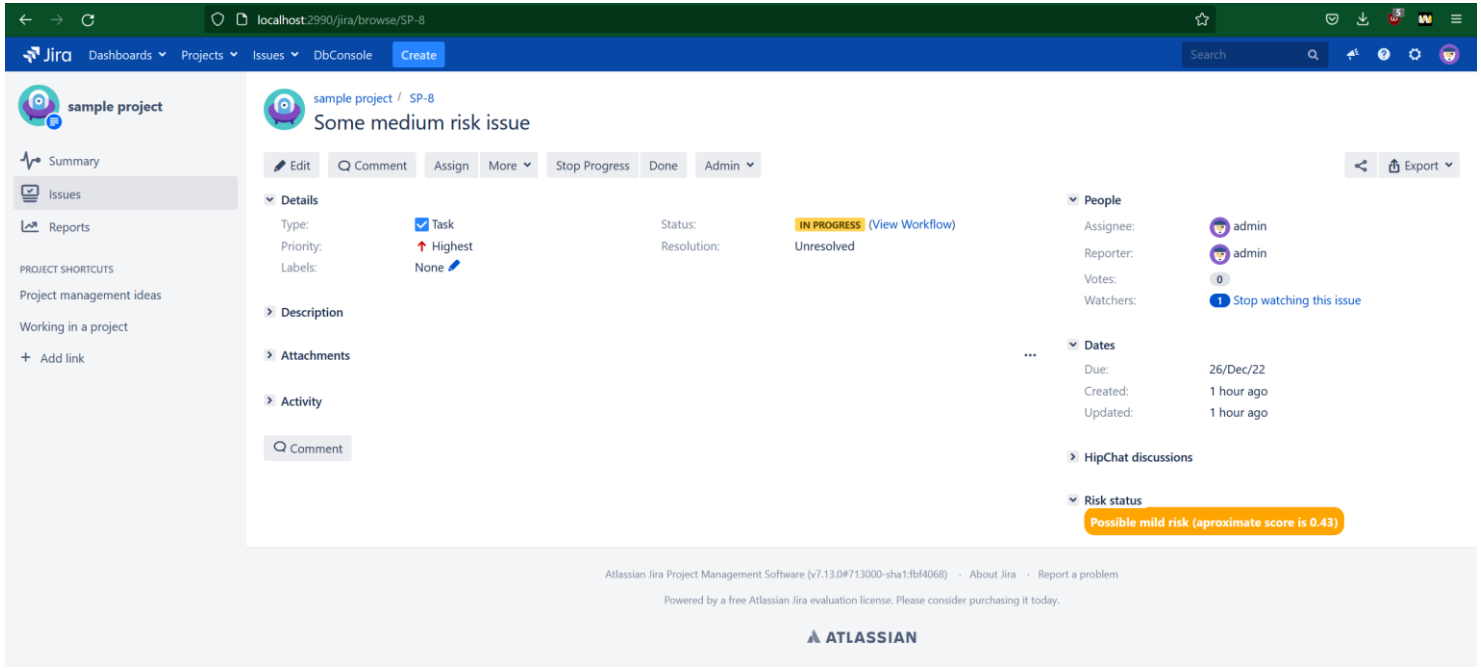
Архітектура програмної реалізації модулю навчання



Архітектура програмної реалізації модулю ідентифікації



Приклад роботи програмної реалізації



Наукова новизна

Вперше запропоновано модифікацію методу випадкового лісу для врахування даних на основі дат, застосовуючи коефіцієнт відхилення ймовірності ризику, отриманий процедурою прогнозування даних на основі часових рядів Prophet. Отриманий метод підвищив характеристики релевантності влучності, повноти та F-міри на 0,01-0,02 відносно до класичного методу випадкового лісу.

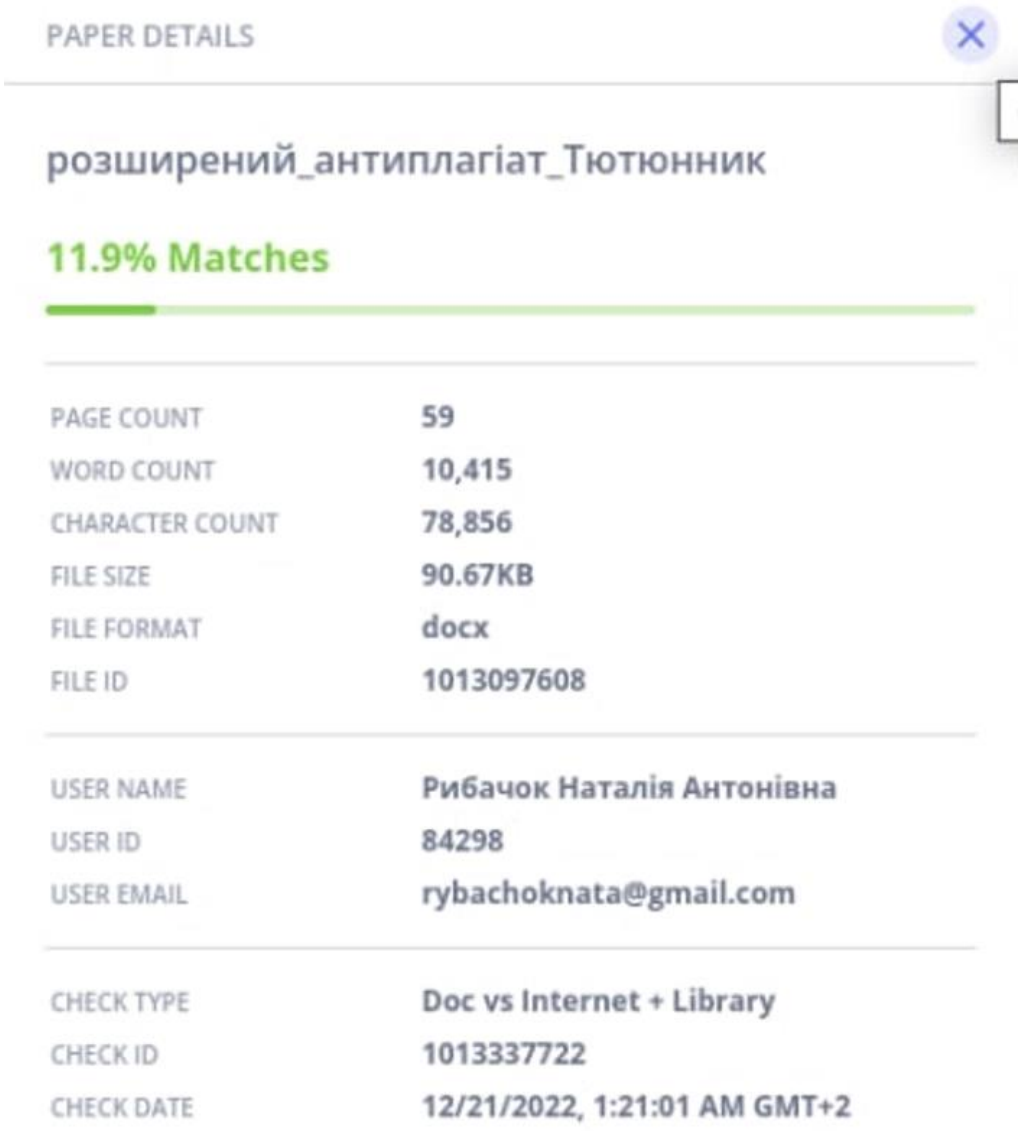
Апробація

Дана дисертація була представлена на науковій конференції магістрантів та аспірантів «Прикладна математика та комп’ютинг ПМК-2022» (Київ, 16-18 листопада 2022 р.)

Рибачок Н.А., Тютюнник П.Б. Метод визначення можливих ризиків та затримок в AGILE проектах. Прикладна математика та комп’ютинг. ПМК – 2022. Збірник тез доповідей XV наукової конференції магістрантів та аспірантів. Київ, 16-18 листопада, 2022. pp. 488-492.

За результатами роботи написана стаття “SOFTWARE METHOD OF IDENTIFYING RISKS AND DELAYS IN AGILE PROJECTS”, яка прийнята до опублікування у журнал категорії Б “Control Systems and Computers” і планується до опублікування в №1 (301), 2023 року.

Унікальність роботи


PAPER DETAILS

розширений_антиплагіат_Тютюнник

11.9% Matches

PAGE COUNT	59
WORD COUNT	10,415
CHARACTER COUNT	78,856
FILE SIZE	90.67KB
FILE FORMAT	docx
FILE ID	1013097608
USER NAME	Рибачок Наталія Антонівна
USER ID	84298
USER EMAIL	rybachoknata@gmail.com
CHECK TYPE	Doc vs Internet + Library
CHECK ID	1013337722
CHECK DATE	12/21/2022, 1:21:01 AM GMT+2

ВИСНОВКИ

1. Досліджено існуючі програмні методи ідентифікації ризиків та затримок у Agile-проєктах.

2. Розроблено модифікований метод ідентифікації ризиків та затримок у Agile-проєктах на основі методу випадкового лісу та процедури Prophet.

3. Розроблено програмне забезпечення, що реалізує запропонований метод.

4. Зібрано інформацію про проблеми Agile-проєктів з відкритим програмним кодом та сформовано набори данних зі знайдених даних.

5. Проаналізовано характеристики релевантності запропонованого методу у порівнянні з існуючими методами.

Дякую за увагу!