Хмельницький національний університет
Факультет інформаційних технологій
Кафедра комп'ютерної інженерії та інформаційних систем

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

Галузь знань 12 – Інформаційні технології

Спеціальність 123 –Комп'ютерна інженерія

на тему «Метод та система розподілу та оцінювання задач в процесі розробки програмного забезпечення комп'ютерних систем»

КвРКІП. 170283.21.01.01 ПЗ

Виконав: студент 2 курсу, група КІ2м-21-1 _____ Окрушко Д.В.
Підпис Ініціали, прізвище

Керівник доцент кафедри _____ Каштальян А.С.
Науковий ступінь, вчене звання Підпис Ініціали, прізвище

До захисту допускаю:
Зав. кафедри КІІС, д.т.н., проф.
Т.О. Говорущенко
04 05 2023 р.

Хмельницький, 2023

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ ТА ІНФОРМАЦІЙНИХ СИСТЕМ

Освітній рівень МАГІСТР

Галузь знань 12 ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ

Спеціальність 123 КОМП'ЮТЕРНА ІНЖЕНЕРІЯ

Освітня програма ОСВІТНЬО-НАУКОВА ПРОГРАМА «КОМП'ЮТЕРНА ІНЖЕНЕРІЯ ТА ПРОГРАМУВАННЯ»

ЗАТВЕРДЖУЮ

Зав. кафедри Т.О.Говорущенко

" 01 " 09 2022 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ МАГІСТРА

Окрушко Дмитру Віталійовичу
Прізвище, ім'я, по батькові студента

1. Тема проекту (роботи) Метод та система розподілу та оцінювання задач в процесі розробки програмного забезпечення комп'ютерних систем
Керівник проекту (роботи) Каштальян А.С. доцент кафедри.
Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 09.01.2023 р. № 1

2. Строк подання студентом проекту (роботи) на кафедру 01.05.2023 р.

3. Вихідні дані до проекту (роботи) Завдання на дипломне проектування

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)
Аналіз відомих моделей, методів розподілення та класифікації задач у галузі.
Моделі для вирішення розподілення та класифікації задач
Алгоритми та технології для вирішення задач
Реалізація програмного забезпечення для вирішення задач.

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень)

6. Консультанти розділів кваліфікаційної роботи магістра

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Лисенко С.М., професор кафедри КІС		
Антиплагіат	Нічепорук А.О., доцент кафедри КІС		

7. Дата видачі завдання «_06_»_____09_____2022р.

КАЛЕНДАРНИЙ ПЛАН

№з/п	Назва етапів (розділів) кваліфікаційної роботи магістра	Термін виконання етапів проекту (роботи)	Примітка
1	Вибір напряму дослідження та узгодження тематики КвРМ з керівником	05.09.2022	виконано
2	Ознайомлення з предметною областю; формулювання мети та задач дослідження; визначення об'єкта та предмета дослідження	05.10.2022	виконано
3	Робота над розділом 1 – аналіз відомих моделей, за темою; постановка задачі	05.11.2022	виконано
4	Робота над розділом 2 – розробка моделей для вирішення поставленої задачі	05.12.2022	виконано
5	Робота над науковою статтею	05.01.2023	виконано
6	Робота над розділом 3 – розробка алгоритмів для вирішення поставленої задачі	15.02.2022	виконано
7	Робота над розділом 4 – реалізація ПЗ для вирішення поставленої задачі, експериментальна частина	05.04.2023	виконано
8	Оформлення пояснювальної записки згідно вимог	15.04.2023	виконано
9	Попередній захист ДРМ	18.04.2023	виконано
10	Захист ДРМ на засіданні ЕК	До 10.05.2023	

Студент Підпис Д.В. Окрушко
Ініціали, прізвище

Керівник роботи Підпис А.С. Капітальян
Ініціали, прізвище

РЕФЕРАТ

Тема кваліфікаційної роботи магістра: Метод та система розподілу та оцінювання задач в процесі розробки програмного забезпечення.

Автор роботи: Окрушко Д.В. студент групи КІ2м-21-1.

Керівник роботи: доцент кафедри, Каштальян А.С.

Пояснювальна записка: 90 с., 31 рис., 3 табл., 3 дод., 86 джерел.

ДЕРЕВА ПРИНЯТТЯ РІШЕНЬ, ЛІНІЙНЕ ПРОГРАМУВАННЯ, РОЗПОДІЛЕННЯ ЗАДАЧ, КЛАСИФІКАЦІЯ ЗАДАЧ, API, СППР, SCRUM, AGILE.

Об’єктом дослідження є процес автоматичного розподілення та класифікації задач під час розробки програмного забезпечення.

Предметом дослідження є метод та засоби для автоматичного розподілення та класифікації задач під час розробки програмного забезпечення

Метою кваліфікаційної роботи магістра є автоматизація та підвищення ефективності процесу розподілення та класифікації задач.

Для розв’язання поставлених задач використовувалися методи які були розроблені на основі алгоритмів дерев ухвалення рішень та методів лінійного програмування. На основі проведених досліджень розроблена архітектура і компоненти програмного забезпечення які виконують автоматичне розподілення задач між учасниками рейтингів та виконує класифікацію поставлених задач. Система також генерує навчальний дата сет, на базі якого можна дослідити етапи навчання алгоритму та візуалізувати отриманні дані.

Наукова новизна отриманих результатів:

− удосконалено метод для визначення найкращого кандидата на виконання завдання, який забезпечує, створення системи рейтингів базуючись на оцінках як виставили інші користувач, враховує завантаженість людини, також враховує класифікацію задач на наявність тегів preffered чи not preffered.;

− розроблено метод для визначення класифікації завдання, який забезпечує пошук групи тегів як належать до конкретного завдання структури завдання з новими класифікаційними тегами;

Практична значимість отриманих результатів полягає у

− розробці інформаційної системи для визначення найкращого кандидата на виконання завдання та виконувати класифікацію завдань, за допомогою визначених тегів, з використанням стеку MERN і Firebase у програмної платформи для проектованої системи.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

FB – Firebase

ШІ – штучний інтелект

БД - база даних

СУБД – система управління базами даних.

СППР – системи підтримки і прийняття рішень

ГА - генетичний алгоритм

ПЗ - програмне забезпечення

СПЗ – системне програмне забезпечення

ДПР - дерева прийняття рішень

ЛП - лінійне програмування

ВСТУП

На сьогоднішній день розробка програмного забезпечення для різноманітних комп'ютерних систем – це складний процес, який вимагає ретельного планування, дотримання планування і виконання. Це включає в себе створення різноманітних задач, від проектування та кодування до тестування, розгортання, зустрічей із замовниками тощо. Щоб забезпечити успішне виконання усіх цих поставлених задач, дуже важливо мати ефективний метод, систему яка буде відповідати за розподілення цих завдань між виконавцями та виконувати оцінку цих задач.

Оцінювання задач у свою чергу для різних видів та типів завдань має свої певні характеристики та особливості. Це пов'язано насамперед із складністю функціювання такої системи та створенням певних видів архітектури комп'ютерних систем. Було досліджено роботу [1] , в якій було представлено багато методів для розподілу та оцінювання задач в процесі розробки програмного забезпечення. Але описані методи та алгоритми мають свої особливості, що не дозволяють автоматизувати систему розподілу та класифікації завдань, тому ці підходи мають бути оптимізовані і покращені наскільки це можливо. Також описані [3] засоби розподілення і класифікації задачі були розроблені із врахування такого фактору як безпека. Рівень захисту даних є важливим фактором, але на практиці цим фактором нехтують, або перекладають відповідальність за безпеку даних на інші сервіси. Існують і інші методи які враховують стійкість тих чи інших елементів системи розподілення задач під час великих навантажень на систему. Під час відмови тих чи інших вузлів системи, процес прогнозування, розподілення і класифікації завдання, повинні передатись на вузли для повторного виконання [2]. Вище описаний процес сильно впливає на швидкодію усієї системи загалом. Але такий підхід є не дуже зручним для кінцевого користувача Також існують механізми які використовуються для системи розподілення та класифікації задач, які полягають у тому що мають враховуватись параметри обміну даними всередині системи, а також де ці дані розміщенні і наскільки великий об'єм даних доступний для навчання. Тому було прийнято рішення використовувати один механізм який

поєднує та викликає один за одним потрібні механізми із відповідними комунікаціями як і в середині системи так і із зовнішніми викликами на різні ресурси для прискорення або уточнення даних. Для системи розподілення, оцінювання та класифікації задач були розроблені різні моделі. Моделі можуть використовувати як базові принципи, так і складатися із інших моделей. Час затрачений для розподілення задачі або її класифікації може бути великим, що має негативний вплив на фактор часу. Тому найкращими варіантами вважають рішення які балансують на межі між двома параметрами це – швидкість класифікації, тобто час, та оптимальна точність. На даний момент як розробники так і вчені ведуть активні дослідження над оптимальним алгоритмом який поєднає в собі ці два фактори, де основний уклін йде на хмарні технології, оскільки можна використати необхідну обчислювальну потужність сервера. Прогнозується що кожної хвилини система повинна опрацювати близько ста задач. А це означає що потрібно мати ефективне та гнучке управління ними.

Під час планування розподіленням задач та управліннями даними всередині системи потрібно мати чітку та визначену структуру і визначені критерії, які визначаються системою, алгоритмом, методом, моделюю. Системи класифікації задач яка використовує в основі технології хмарних обчислень має низку переваг та властивостей які у свою чергу з одного боку дозволяють спростити навантаження на обчислювальну частину а з іншого боку ускладнює взаємодію, інтеграцію та виявленню помилок під час керування процесом розподіленням та класифікацією задач. Також до основних властивостей для такого роду відносяться: різноманітність архітектури, швидка зміна процесів, наявність великої кількості категорій та класифікаторів, не повний опис завдання, допустимість помилок введення даних користувачем, можливість користувача використовувати можливості обчислень не за призначенням тощо. Отож на сьогодні відомі алгоритми, моделі, методи та системи які займаються розподілом та обчисленням задач в процесі розробки програмного забезпечення комп’ютерних системи не є досконалими та не вирішують вище описаних проблем.

1 АНАЛІЗ ВІДОМИХ МОДЕЛЕЙ, МЕТОДІВ ТА ЗАСОБІВ У ГАЛУЗІ

1.1 Аналіз особливостей та дослідження моделей системи розподілення та класифікації задач

Аналіз та дослідження особливостей системи розподілення та класифікації задач дуже залежить від ступеня засвоєння та можливості інтеграції його в інші наявні системи. А також важливо дослідити які саме методи та підходи використовують менеджери на проектах для того щоб можна було автоматизувати цей процес на скільки це можливо.

Перші підходи почали з’являтися ще у кінці 1960-х років. Так як індустрія програмного забезпечення та і загалом всього IT швидко розвивалась великі IT компанії побачили потенціал у створенні програмного забезпечення. Менеджери компаній використовували популярну на той час водоспадну модель при розробці програмного забезпечення та інших своїх продуктів. Waterfall (або водоспад) - це модель, яка використовується під час розробки проекту. Це була одна із найперших моделей яку почали використовувати у процесі створення програмного забезпечення. Модель Waterfall надає розробникам і менеджерам покроковий план роботи, де перш ніж переходити до наступного кроку потрібно виконати всі попередні, оскільки вони є блокуючими для наступних кроків. Це дуже схожу до візуалізації у Canban, де менеджери і команди розробників можуть спостерігати за всім процесом наявних та виконаних завдань із можливістю спостерігати у реальному часі які завдання блокують наступний етап і які задачі є більш в пріоритеті. Waterfall реалізується як модель послідовного життєвого циклу. Waterfall стартує із збору команди та документування усіх потрібних вимог. Далі відбувається проектування рішення, яке буде вирішувати поставлені задачі. Потім це все тестується, покривається тестами і віддається замовнику. Але важливе уточнення що розробники мають закінчити крок перед ти як починати новий.

Модель Waterfall має шість фаз, які зображені на рисунку 1.1. Де кожна фаза була розроблена у такий спосіб щоб було зрозуміло коли можна починати наступний крок.

Під час першої фази команда лише отримує всі вимоги від клієнта, робить базову документацію. Вони повинні бути детально досліджені з урахуванням чітко визначених кінцевих цілей.

На другій фазі, проектування програмного забезпечення команда розробників створює усю необхідну базову архітектуру. Після того як визначили усю необхідну базову структуру та потрібне системне програмне забезпечення розробники приступають до основної фази написання коду.

Після того як створена базова структура, визначені необхідні сервіси і отримані доступи до сторонніх сервісів, розробники починають головну фазу імплементація коду, де кожен модуль розробляється окремо, який потім обов'язково тестується та покривається тестами. Також кожен модуль обов'язково має бути повністю працездатним перед наступною фазою.

Під час фази тестування та інтеграції всі створені модулі об'єднуються у одну систему, яка потім і стане фінальним продуктом. Розробники або спеціальна команда із тестування мають протестувати усю систему, і виправити усі знайдені помилки які знайшли.

На наступній фазі доставки і розгортання розробники мають встановити програму або мають виконати адаптацію програми у стан, який зможуть використовувати кінцеві користувачі. Зазвичай під час цієї фази розгортаються автоматичні тести на серверах. І лише після вище описаних пунктів системи потрапляє у так званий стан який використовують кінцеві користувачі.

Останньою фазою є запуск програмного забезпечення, де головним завданням є підтримка продукту або внесення нових побажань клієнта а також виправлення помилок які не були виявленні під час тестування.

Рисунок 1.1 – Водоспадна (Waterfall) модель

Водоспадна модель дуже проста у використані що дає їй ряд переваг, таких як:

1. Чітко заданий поділ фаз розробки.
2. Пропонує велику кількість документації. Усі етапи мають бути задокументовані.
3. Команди знають обсяг який необхідно буде виконати.
4. Присутність клієнтів лише на початковому та кінцевому етапах.

З часом стало зрозуміло що модель також має і свої мінуси:

1. Оскільки вимоги можуть бути не завжди зрозумілі немає достатньої гнучкості яка дозволяє змінювати кінцевий результат у простий спосіб.
2. Жорсткий поділ на фази розробки.

Цих два простих обмеження призводили до збоїв під час розробки і розподілення задач, а причини були такі:

1. Неточні оцінки проекту.
2. Наявність ризиків які важко контролювати.
3. Наявність не точно визначених вимог.
4. Зазвичай використання старих або не підходящих технологій написання коду.
5. Також можливо через надмірну складність проекту.
6. Наявність комерційного тиску.

Наявність такої кількості недоліків вимагає створення нових підходів до розробки програмного забезпечення та систем які допоможуть контролювати процес та підходи які б мали більшу гнучкість. Нові методи почали з’являтися ближче до кінця 1980-х і протягом 1990-ч років. Ці методи використовували поетапний та ітераційних підхід до розробки програмного забезпечення. Це було зроблено для того щоб мати більш швидку відповідь від замовника на кожній ітерації розробки програмного забезпечення, заохочуючи замовника приймати участь у кожній ітерації і вносити зміни більш швидко, що давало таким системи значну перевагу у гнучкості у порівнянні зі Waterfall. Що потім призвело до об’єднання найбільш гнучких та стрімко зростаючих методів по популярності серед розробників та менеджерів у одну велику і гнучку систему, яка у свою створила новий напрямок у системі розподілення та менеджменту задач серед розробників.

1.2 Аналіз методів Agile для розробки та розподілення задач

У 2001 році з’явився такий метод як Agile Alliance. До його складу входять найбільш гнучкі та динамічні методи. Також був створений документ який

визначав базові концепції підходу Agile і називався він Agile Software Development (Agile Alliance) [4-7]. Було створено 4 основні концепції які визначали базові цінності Agile, і ці цінності повинні використовувати всі Agile-технології рисунок 1.2. Наявні відмінності між двома моделями які були проаналізовані та досліджені наведено у таблиці 1.1. Як показує практика загальна кількість невдалих проектів, або проектів які оскаржуються є достатньо великою.

Рисунок 1.2 – 4 базові цінності Agile-методу

У роботі [8, 9] було надано інформацію, що лише майже 40 відсотків усіх наявних проектів є успішними. Близько 45% були оскаржені по різним причинам (немає усіх запланованих функції і можливостей програмного забезпечення, було перевищено виділений бюджет, поставлені задачі були виконані не вчасно, тощо). А 15% відсотків були взагалі невдалими(були виконані задачі, але системне

програмне забезпечення так і не потрапило на ринок, або взагалі скасовано замовлення на системне програмне забезпечення). Також було зазначено що з 2004 року було поступове збільшення усіх успішних проектів з 30 відсотків, в середньому на 5 відсотків.

Таблиця 1.1 – Порівняння двох методів Agile та Waterfall

Agile	Waterfall
Планування на короткі періоди із можливістю динамічним доповненням потрібних вимог.	Планування займає багато часу, оскільки цей процес вимагає довгострокового планування, і вимоги не можуть бути зміненні під час розробки програмного забезпечення затвердженим по плану виконання і наявної побудованої інфраструктури в середині системи.
Усі члени команди відіграють однакову роль у команді.	Усі члени команди слідують вибудованій ієрархічні структурі у організації.
На всіх етапах розробки відбувається постійна комунікація із клієнтом, що в свою чергу дозволяє отримати вчасний відгук від клієнта і зрозуміти чи правильно все було зроблено і зрозуміло.	Повністю незалежний від спілкування із замовником.
Завдяки тому що відбувається послідовне тестування відбувається і постійний контроль якості програмного забезпечення.	Через пізнє планування може відставати контроль якості, оскільки тестування займає багато часу. Також виправлення помилок має бути наявним у системі

Кінець таблиці 1.1 – Порівняння двох методів Agile та Waterfall

Можна зручно вимірювати і слідкувати за прогресом виконання продукту завдяки вимірюванням у спринті або одиницях.	Увесь прогрес вимірюється за реалізованими кроками.
Терміни виконання програмного забезпечення можна зручно адаптувати якщо під час одного із спринту з'являються додаткові вимоги до кінцевого продукту.	Терміни можна легко визначити, оскільки внесення нових змін не прогнозується і якщо не буде не передбачуваних збоїв у системі.
В залежності від проектів їх можуть ділити на різні категорії такі як : складні, поточні або довгострокові, тощо	Найкращий підхід для проектів які не мають великим розмірів, і усі кроки можна спланувати або вони є передбачуваними і на них можна вплинути. Обов'язково має бути відомо усі вимоги.

У IT сфері існує поширена думка що, команди які використовують у своєму підході Agile значно збільшують шанси ймовірного успіху програмного забезпечення. Попри те що це не завжди підтверджено наявними емпіричними даними, але технологія Agile тривалий час сприймалась як методологія, що зробить великий прорив в розробці програмного забезпечення для проектів різних масштабів. Нещодавнє дослідження в роботах [14, 15, 16] що досліджували успіх Agile проектів показує що можливість такої думки може бути доведена на наукових даних. В роботах було показано кількісне дослідження, для того щоб перевірити чи впливає на процент успішності продуктів які використовували гнучкі методи Agile під час розробки своїх продуктів. Вони помітили ознаки того, що використання методів гнучкого планування у процесі розробки програмного забезпечення має значно більший зареєстрований показник успішності продуктів. Оцінювання і

показ результатів було проведено для трьох категорій успіху: загальний успіх проекту, успіх зацікавлених сторін та ефективність.

Agile проекти добре відомі своїм гнучким процес розробки програмного забезпечення в порівнянні з традиційними проектами. Сам процес складається з великого набору практик, який у свою чергу описує набір процедур, які потім використовують команди розробників для того щоб досягнути кінцеві цілі проекту.

Для проектів в сфері IT і метою яких є створення якісного програмного забезпечення, гнучкий підхід допомагає визначити масштаб і розміри проекту на всіх його стадіях, включаючи і останню стадію релізу. Розмір, потрібний час, а також вартість та якість є досить важливими критеріями при розгляді успіху програмного продукту.

Також не варто забувати що для різних підходів, необхідно мати різні реалізації. Коли потрібно обрати метод також потрібно врахувати чи підходить обраний метод для виконуваного проекту.

Метод який описує практики або принципи у вигляді ролей, артефактів та подій – це Scrum. Також існує двадцять чотири інженерних практик, за допомогою яких описують екстремальне програмування.

В свою чергу Kanban описує п’ять базових принципів[23]. Також Kanban можна вважати як об’єднання найкращих практик, принципів та цінностей.

Scrum – це структура яка є, як ітераційною так, і інкрементною. Вона була створена спеціально для проектів програмного забезпечення які беруть за основу гнучку розробку продукту. Scrum – був описаний ще в 1995 році, але коли був заснований Agile Alliance він почав набирати популярність у 2001 році.

У Scrum є лише три ролі, це: Член команди, Власник продукту, Scrum-master.

Як відомо замовник або ж власник продукту визначає потрібні вимоги і контролює процес розробки продукту. Процес Scrum організовує Scrum master. Розробники, дизайнери, менеджери, тестувальники є також членами команди і беруть участь у процесі Scrum.

У процесі Scrum є чотири події: Огляд спринту, Щоденний і щотижневий мітинг, Перегляд спринту, Планування спринту.

Будь яка подія, яка має певну тривалість і обмеженість у часі, використовується для постійного потоку різної інформації між усіма зацікавленими сторонами.

Спринт – це цикл довжиною в два або чотири тижні, при якому команда працює над заданими завданнями.

Артефакти які є найбільш важливими [24]: Збільшення розмірів продукту, Відставання від запланованого спринту, Відставання самого продукту.

Вимоги які будуть розроблені в майбутньому називаються backlog. У пункті відставання від спринту є вимоги, які опрацьовують обов'язково в рамках поточного спринту.

Під час процесу планування спринту, команда і замовник вирішують зазвичай які саме завдання мають увійти у план виконання для спринту.

Під час спринту розробники працюють над тим щоб виконати усі поставлені задачі у плані спринту. Також проводять щоденні зустрічі, для того щоб, розповісти хто що зробив, і хто що планує робити, також під час цих зустрічей обговорюють потенційні проблеми і шляхи їх вирішення.

Kanban представляє собою скоріше мислення, ніж методологію. Про це свідчить той факт, що Kanban не надає багато правил , як слідувати процесу, а скоріше фокусується на правильному відношенню до тих чи інших речей.

Kanban пропонує базовий набір правил для різних аспектах в не залежності від розміру проекту:

1. Зробіть так щоб виконуваний прогрес був явним.
2. Спрямовуйте та керуйте процесом розробки.
3. Потрібно візуалізувати робочий процес.
4. Покращуйте командну роботу.

Кожен обраний елемент для розробки знаходиться в резерві із різними пріоритетами. У програмній реалізації Kanban це дошка які містить декілька стовпців, де кожен стовпець відповідає за статус.

Кожен елемент Kanban знаходить в одному із стовпців і має можливість змінити своє розміщення із зліва на право під час роботи із ним. Також можливо

встановити обмеження для кожного стовпця із вказанням скільки елементів може містити колонка в певний момент часу (тобто під час одного спринту). Якщо існує багато елементів в одній колонці і вони блокують увесь потік, команда має сфокусуватись над вирішенням задач в цих колонках і відновленням робочого потоку.

Також існує програмно орієнтований метод швидкої розробки який називається – екстремальне програмування. В цьому підході існує п’ять основних цінностей які призначені для підвищення продуктивності та покращення якості коду.

Як і всі системи Kanban має ряд недоліків:

1. Kanban може загальмувати процес в один момент. Для того щоб усе чітко працювало, усі процеси, повинні бути добре налаштовані і кожен у команді має знати, що входить у його роботу. Якщо співробітники не впевнені в тому, що відноситься до їхньої роботи та зони відповідальності, завдання можуть перестати виконуватись, а це в свою чергу спричинить ефект доміно.
2. Kanban вимагає постійно наявності завдань. Якщо завдань не буде, почнеться процес руйнування.
3. Немає часових обмежень. В Kanban немає термінів, для виконання завдання, для виконання завдання дається стільки часу, скільки потрібно. Це може створити проблеми з дедлайнами проекту.
4. Пріорітетність задач втрачається . Оскільки співробітники самі беруть завдання в роботу, вони можуть обрати ті, які мають низький пріоритет для бізнесу.

Також існує декілька різних комбінацій, або ж гібридів. Найпопулярніші із них це Scrumban and Scrum/XP.

Scrum/XP – це комбінація правил і підходів разом із Scrum і екстремального програмування. Цей підхід надає середовище, для розробки програмного продукту у такий спосіб, який є більш зручним для бізнесу.

Ці п’ять основних практик: Ітерація, Прирости, Появи, Самоорганізація, Співпраця.

Scrumban – це підхід який поєднує Scrum і Kanban. По факту це поєднання подій і артефактів Scrum і Kanban дошки з відслідковуванням прогресу. Основна відмінність це відсутність чітко визначених проміжків часу. Повна відсутність спринтів, натомість весь робочий процес безперервний.

Існують також практики Agile які також використовують під час розробки програмного забезпечення. До цих практик входять: Інтерактивна розробка, Визначення готовності, Діаграма згорання, Щоденні мітинги, Визначення готовності, Існування беклогу, Використання інструментального дизайну.

Часто можна зустріти думку, що Agile-команди мають багатофункціональні здібності та добре само організовуються. Але це означає що команда несе відповідальність в залежності від своїх обмежень та можуть самостійно вирішити проблеми які з’являються під час розробки проекту.

Agile і Scrum теж мають ряд недоліків:

1. Потрібно добре структурувати робочий процес. В Scrum більше зустрічей і конкретних процесів, ніж у Kanban, тому його не так легко підлаштовувати під різні бізнес-сценарії.
2. Наявність конкретних ролей (Scrum-майстер, власник продукту тощо) та процесів (щоденні зустрічі, спринт-огляди) означає, що команді доведеться звикати та підлаштовуватись до цього.
3. Забагато нарад. Кожен спринт досить короткий, але потребує декількох сеансів обговорення.
4. Scrum вимагає часу на впровадження. Знадобиться кілька спринтів, щоб співробітники зрозуміли, як це працює.
5. Необхідно правильно планувати робоче навантаження. Якщо робота не буде належним чином розбита на невеликі компоненти або її буде забагато для одного спринту, виникнуть затримки в проекті.

1.3 Аналіз системи та підходів систем підтримки прийняття рішень

СППР (або ж Системи підтримки прийняття рішень) – це такі інформаційні системи, що допомагають в управлянні підприємством так процесам які сприяють цьому [27]. Така інформаційна система допомагає керувати бізнесом та виконуваними процесами. Може використовуватись у організаціях як високого так і середнього рівня. Ця система накопичує інформацію та аналізує великі обсяги не структурованих даних, які потім можуть допомогти у вирішенні проблем коли потрібно прийняти рішення. Системи підтримки та прийняття рішень можуть створювати детальні звіти завдяки накопиченню великих обсягів даних та їх аналізу.

Для організації своєї роботи система підтримки прийняття рішень, система керується відділами, або ж зонами відповідальності. Відділ який збирає дані може згенерувати звіти, які потім будуть використовувати менеджери для прийняття рішень. В основному головна ціль системи прийняття рішень це прогнозування виконання робіт, для різних сфер менеджменту.

Такі системи можуть використовуватись у багатьох сферах знань, включаючи в цей список і управління проектами в сфері IT технологій і управління медичними або і іншими послугами. Головним застосування системи підтримки та прийняття рішень є генерація звітів у режимі реального часу, що може бути корисним для організацій, в яких в управлінні в основі лежить підхід “Вчасно та точно” або ж JIT.

JIT – це підхід який вимагає дані про наявні ресурси (включаючи сюди людей) у режимі реального часу, щоб зробити звіт вчасно та точно. Це робиться для того щоб уникнути ефекту доміно та затримкам у виробництві. Реалізація для IT проекту полягає в управлінні задачами які мають різні пріоритети.

Оскільки зміна пріоритетів впливає на загальну зміну послідовностей виконання задач в розробці проекту.

Це допомагає менеджерам і розробникам можливість врахувати терміни завершення конкретних задач і вплив на загальний термін виконання проекту і всіх поставлених задач.

СППР – складається з трьох базових складників. Такі як: Модуль підтримки на основі моделі, Модуль підтримки на основі знань, Наявний інтерфейс для комунікації із користувачем рисунок 1.3[28].

Окремий модуль підтримки рішень на базі знань, може використовувати інформацію що надходить із різних джерел. Наприклад внутрішніх джерел і зовнішніх джерел. До внутрішніх можна віднести інформацію із наявних транзакцій, а до зовнішніх базу даних. На основі отриманої інформації генеруються знання для об’єкту управління.

А для модулю підтримки рішень що базується на основі моделі, то в своїй основі він зберігає різні моделі. Ці моделі потім використовуються для прийняття рішень менеджерами. Зазвичай ці моделі використовують для рішень фінансового сектору організації, процесу прогнозування попиту на послугу чи товар. Можуть бути використані також для планування різних задач.

Рисунок 1.3 – Підтримка рішень на основі знань та моделі в системі підтримки та прийняття рішень

Основна відмінність між цими двома модулями у тому, що в модулі на основі бази знань використовується знання про існуючі процеси управління, а для модуля

який базується на основі моделей використовується інформація про процеси на об’єкті управління.

Модуль інтерфейсу для комунікації із користувачем включає в себе набір інструментів який допоможе користувачеві використовувати СППР. У системах СППР зазвичай використовується як текстовий інтерфейс так і графічний, що є досить зручним у процесі використання.

СППР мають ряд переваг, які дозволяють менеджерам удосконалити процес управління:

1. Враховуючи що система підтримки і прийняття рішень може збирати і аналізувати дані в реальному часі, то підвищується ефективність з прийнятті рішень.
2. Враховуючи що систему підтримки і прийняття рішень потрібно знати як впровадити в робочий процес, це розвиває навички решти персоналу.
3. СППР допомагає автоматизувати однорідні процеси в управлінні, і це дає змогу керівникові витрачати більше часу на прийняття важливих рішень.
4. СППР допомагає покращити комунікацію в середині персоналу.

Але СППР також має ряд недоліків. Такі як:

1. Достатньо великі витрати для впровадження СППР. Висока вартість не дозволяє використовувати такі системи в невеликих компаніях.
2. Оскільки система може бути адаптована і впроваджена в усі процеси, це може призвести до залежності персоналу від СППР. Зазвичай люди настільки покладаються на такі системи що віддають важливі рішення саме цим системам.
3. Оскільки система працює в усіх напрямках, то система може надавати велику кількість інформації яка не зовсім потрібна саме зараз, це призводить до інформаційного перевантаження людей.

4. У працівників нижчого рівня впровадження системи СППР може викликати страх що їх скоро повністю замінять. Вони бояться втратити роботу через впровадження у виробничі процеси новітні технології.

Таблиця 1.2 – Варіанти властивостей для підтримки рішень.

Метод	Властивості
Персональний	Менеджер для прийняття рішень має використати модель конфлікту.
Поведінковий. (процесний)	Менеджер який приймає рішення має обмежити вхідні дані під час прийняття рішення, зосереджуючи свою увагу на більше надійних альтернативах (часто враховуючи обмеження у часі), досліджують ці альтернативи якомога детальніше і потім приймають рішення на основі своїх особистих рішеннях та судженнях.
Раціональний	Менеджер який приймає рішення має мати повну інформацію про об’єкт управління та є повністю об’єктивним у своїх рішеннях.
Гібридний	Менеджер який має прийняти рішення має вийти за існуючі рамки обмежень та різних правил, і має створити більше альтернативних варіантів із використанням наявних ресурсів (наприклад грошей, часу і інших аспектів практичних ситуацій)

На сьогоднішній день існує декілька різних підходів для прийняття рішень за допомогою СППР. Вони реалізують різні варіанти процесів, які потім використовують менеджери для прийняття своїх рішень. Ці рішення можуть бути як частково раціональними так і повністю раціональними. Частково раціональний підхід містить підмножину варіантів а раціональний підхід містить всі варіанти дій. Наявний перелік рішень і властивостей наведено в таблиці 1.2. Один із низ надає раціональне рішення, інші ж у свою чергу надають обмежено раціональні рішення. Концепція раціонального підходу визначає, що люди які керують організацією є повністю об'єктивними, вони приймають рішення маючи при цьому усю інформацію про організацію [29].Такий підхід має як і сильні сторони так і слабкі. Сильні сторони полягають у тому що особа яка має прийняти рішення, має провести глибокий аналіз усіх варіантів, його рішення має бути логічним та послідовним, без впливу емоцій, без тиску, без особистого ставлення.

Слабкі ж сторони це людина може не завжди володіти усією необхідною та повною інформацією. Зазвичай людина стикається із фінансовими або ж часовими обмеженнями. Тому ці обмеження можуть впливати на здатність усвідомлювати інформацію і не можливо спрогнозувати майбутнє. Також, велика кількість альтернатив може впливати на складність порівняння рішень.

Поведінковий (або ж процесний) підхід говорить про те, що люди, які мають прийняти рішення, повинні діяти із частковою раціональністю. Часткова раціональність базується на тому, що люди не мають і не володіють усією наявною інформацією. Тому їхнє рішення базується на тому, що вони розглядають лише деяку частину цією інформації. Звідси слідує, що наявні рішення не є ідеальними.

Гібридний метод у свою чергу поєднує кроки з умовами поведінкового підходу із кроками раціонального підходу. На меті є створити більш реальний процес прийняття рішень. Для індивідуального прийняття рішення найбільш вдалий підхід базується на основі моделі конфлікту.

Модель конфлікту має такі характеристики:

1. Раціоналізація та прокрастинація є механізмами, які допомагають людям уникати приймати важливі рішення та справлятися із стресом.

2. Модель взаємодіє лише із важливими рішеннями у житті людини
3. Деякі рішення можуть дати не очікуваний результат.
4. Існує ймовірність само реакції під час порівняння існуючих альтернатив із внутрішніми стандартами.
5. Існує ймовірність що особа не завжди розумію на сто відсотків альтернативні варіанти рішень.

Слідуючи конфліктному підходу прийняття рішень можна сказати що особа під час того як зустрічається із якоюсь проблемою, повинна повністю проаналізувати ситуацію, для того щоб знайти зворотній зв'язок (він може бути як позитивний так і негативний), чи існують можливі ризики якщо не змінити тих чи інших параметрів чи дій.

Якщо ж відповідь особи є негативною, людина може продовжувати свою діяльність. Це говорить про те, поточна ситуація не є критичною і якщо нічого не змінити в поточній ситуації ризики можуть взагалі бути відсутніми.

Така ситуація називається нескладна зміна. Вона передбачає внесення змін у вже існуючу поточну діяльність, головне тут щоб зміни не мали різних ризиків.

Також модель конфлікту може розповісти про ідею оборонного уникнення, яке говорить, що якщо змінити процес або групу процесів у поточному стані діяльності допоможе уникнути повторного не внесення змін у систему чи діяльність чи зустрічі із потенційними проблемами, то це потрібно зробити зараз, якщо немає інших варіантів покращити поточний процес.

Усі ці системи мають ряд недоліків:

1. Недостатня прозорість рішень, які приймаються.
2. Недостатня доступність інформації для прийняття рішень.
3. Недостатня здатність до адаптації ситуацій коли змінюються умови.
4. Недостатня здатність до прийняття багатьох рішень одночасно.
5. Недостатня здатність до прийняття рішень за допомогою автоматизованих процесів.

1.4 Висновки. Постановка задачі

В результаті проведеного аналізу виявлено, що існуючі підходи до системи розподілення та класифікації задач мають такі як свої переваги та недоліки. Було проаналізовано багато методів та підходів. Найзручнішим виявився підхід Scrum і Agile. Гібрид цих систем надає перевагу у швидкості реалізації проектів та можливостях контролювати гнучкість розробки під час активної фази розробки і комунікації із замовником. Загалом система розподілення та класифікації задач з використанням agile та scrum дозволяє командам працювати із більшою ефективністю та продуктивністю та швидко адаптуватися до нових вимог та умов.

Також розроблена система має бути легко інтегруватися у робочий процес як система підтримки та прийняття рішень. Яка буде надавати керуючим особам усю необхідну інформацію та моніторингу процесу та можливості адаптувати чи впливати на загальні рішення під час розподілення задач. Найголовніше в системі підтримки та прийнятті рішень для системи розподілу та класифікації задач полягає в повній автоматизації та автономності системи, з наданням усієї актуальності поточного стану задач для менеджерів або керуючих осію в організації.

Постановка задачі:

1. Дослідити поточні процеси розподілення та класифікації задач в команді.
2. Визначити переваги та недоліки поточної системи.
3. Розробити систему розподілення та класифікації задач яку б можна було легко інтегрувати у робочий процес який використовує agile та Scrum.
4. Оцінити ефективність нової системи та провести порівняння із поточною системою.
5. Зробити висновки та запропонувати підходи та варіанти для покращення системи.

2 МОДЕЛІ ДЛЯ ВИРІШЕННЯ ЗАДАЧ

2.1 Порівняльна характеристика існуючих аналогів на ринку

Notion (Рисунок 2.1) – це все-в-одному робочому просторі для нотаток, керування проектами та співпраці. Це веб-застосунок, який дозволяє користувачам створювати та керувати нотатками, завданнями, вікі та базами даних. Іншими словами це суміш усіх популярних форматів, які використовуються для організації роботи: канбан-дошки, задачі які потрібно виконати тощо. Як кажуть самі розробники їхній сервіс покликаний спросити роботу людям завдяки тому що вони можуть поєднати декілька сервісів одночасно. Notion чудово підходить як для особистого використання так і для командного. В ньому зручно працювати із масивом документів, вести облік задач, створювати базу знань.

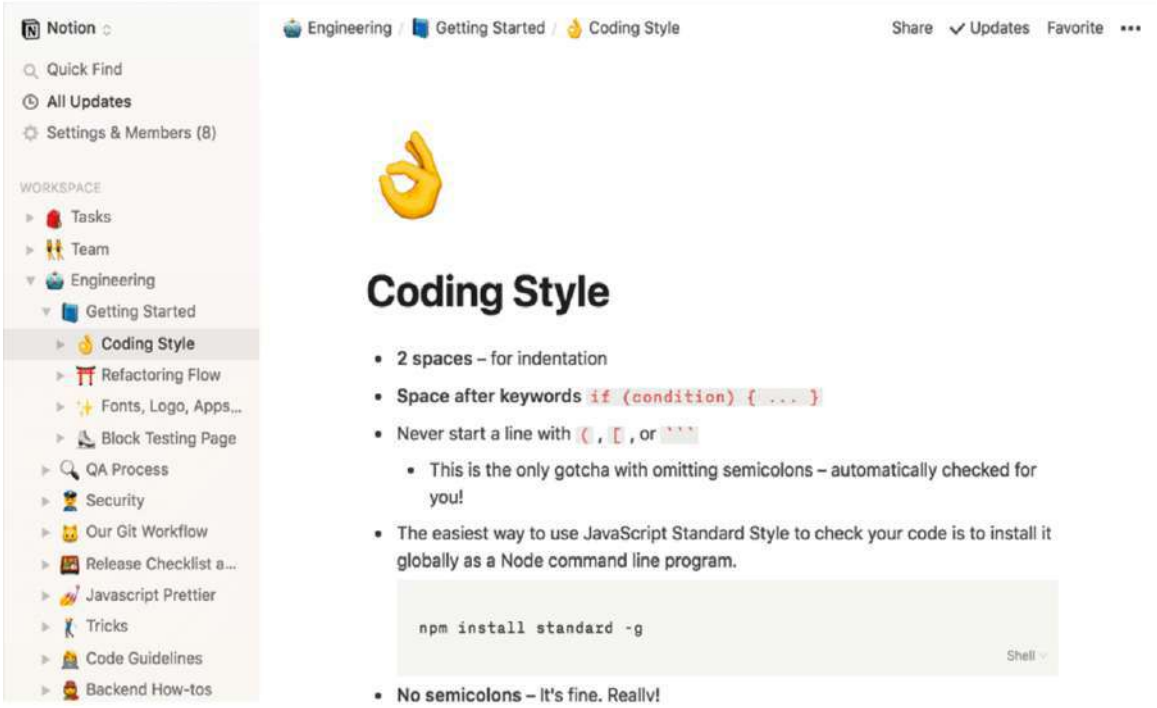


Рисунок 2.1 – Notion інтерфейс

Взаємодія на сторінці починається з білого аркушу. Кожен елемент (абзац тексту, заголовок карточки, чи таблиця) вважається блоком. Порівняння як і в реальному конструкторі, можливо блоки переставляти місцями. Також можна перетягувати по сторінці і поміщати в інші, дублювати, давати їм властивості, додавати в обране.

Переваги використання Notion: Легко використовувати, Notion призначений для того, щоб бути інтуїтивним і зручним, Доступний для IOS & Android, Notion безкоштовний для особистого використання, Наявність помічника Notion AI, Потужна система пошуку .

Недоліки Notion також має. Список недоліків: Немає підтримки для плагінів та додатків сторонніх розробників, Не можливість призначати завдання декільком людям, Відсутність для підтримки для контролю версій, що є досить важливим під час розробки фактором, Немає підтримки для доступу в автономному режимі, Немає підтримки для зовнішніх джерел даних, Немає підтримки для складних формул чи розрахунків.

Також в екосистемі Notion існує Notion AI помічник. Що є великою перевагою на фоні інших конкурентів. Notion AI поєднує в собі функції найпопулярніших нейронних мереж – виправляє та перекладає текст, створює таблиці та пости, відповідає на будь-які запитання користувача. Notion AI корисний для створення нового контенту, але він також може узагальнювати довгі документи, отримати ключові висновки з безладних нотаток, покращуватиме стиль написання користувача тощо. Notion AI не створить нового допису для вашого блогу, який можна одразу публікувати, але допоможе написати чорновий варіант, який надалі користувач може вдосконалювати. Більшість даних Notion AI отримує із інтернету. Із яких саме джерел. Notion AI може стати корисним як генератор ідей. Користувач зможе задати перший варіант, а штучний інтелект продовжить список. Або можна зробити підказку наприклад – “три найкращі документальні фільми” і Notion AI покаже вам потрібний список.

Jira (Рисунок 2.2) – це система управління проектами, яка дозволяє закривати всі завдання Project manager в рамках одного середовища. Від планування до контролю та результату процесів. Jira є комплексом з IT рішень від компанії Atlassian, які об’єднані в Jira Family Products. Jira WM для роботи з бізнес-процесами, Jira SM для побудови сервіс-диску, Jira Software для проектів з розробки програмного забезпечення.

Рисунок 2.2 – Jira як частина екосистеми Atlassian

Jira - це багатогранний помічник для роботи Project Manager, який можна порівняти із конструктором Lego. Без інструкцій ви зможете щось зробити, але при цьому 70% корисних функцій не будуть використані, через що втрачається сенс застосування інструменту.

Знаючи тонкощі роботи Jira завжди можна запропонувати цікаві ініціативи щодо оптимізації процесів, що допоможе заробити додаткові бали у внутрішньому рейтингу компанії. Також є офіційна сертифікація з Jira, яка підвищує конкурентоспроможність фахівця на ринку. Ще одна із причин використовувати всі можливості платформи це можливість отримати джерело ідей для розробки програмного забезпечення. В Jira є що почерпнути для власних проектів. Продумана архітектура, рішення щодо безпеки, реалізація окремих модулів та всього комплексу загалом.

На ринку існує три варіанти сервісу: Jira Cloud, Jira Server, Jira Data center.

Для того щоб почати працювати із Jira потрібно створити новий проект в панелі адміністратора (Рисунок 2.3). Екосистема надає велику кількість наявних шаблонів, також з можливістю обрати методологію роботи із рахуванням специфіки кожної із них. Після чого потрібно обрати тип проекту, для команди чи для компанії. Відмінності між цими типами представлені на Рисунку 2.4.

Після того як було обрано проект система сама створить меню відповідно до обраного шаблону та методології. Для Scrum це буде Backlog, Roadmap, Reports. Також наявна можливість обрати канбан – дошку.

Проєкт під керуванням команди	Проєкт під управлінням компанії
Команда залежить від адміну. Будь-який учасник проєкту може налаштовувати Workflow.	Налаштовується та підтримується адміністраторами Jira.
Налаштування не впливають на інші проєкти.	Налаштування можна розшарити між проєктами.
Спрощене налаштування типів завдань та полів. Лише один тип на рівні підзавдань.	Повний контроль над типами завдань і полями, що налаштовуються.
Спрощені можливості автоматизації.	Повний спектр можливостей автоматизації.
Спрощена модель доступу. Учасник може мати лише одну роль.	Гнучке налаштування доступів та дозволів. Учасник може мати декілька ролей.
Одна дошка у проєкті.	Можливість мати кілька дощок у проєкті.

Рисунок 2.4 – Відмінності між типами проектів

У Jira є можливість обирати доступи до проекту: Open - це повний та відкритий доступ до всього проекту, Limited – будь хто може дивитися завдання в проекті, але лише адміністратор та команда можуть створювати завдання, Private – доступ до перегляду та створення чи редагування мають лише адміністратори та команда проекту.

В Jira є також розділення на ролі: Viewer – користувач має доступ лише для перегляду або можливість залишати коментарі, Member – користувач може створювати та редагувати завдання, Administrator - користувач має повний доступ до всього проекту та завдань.

Головна перевага тут що є можливість створювати нові ролі і надавати їм різного роду доступи та певні привілеї.

В Jira є шість рівнів ієрархії(Рисунок 2.5): Epics, Stories, Tasks, Bugs, Custom types, Sub tasks, та кастомні під типи.

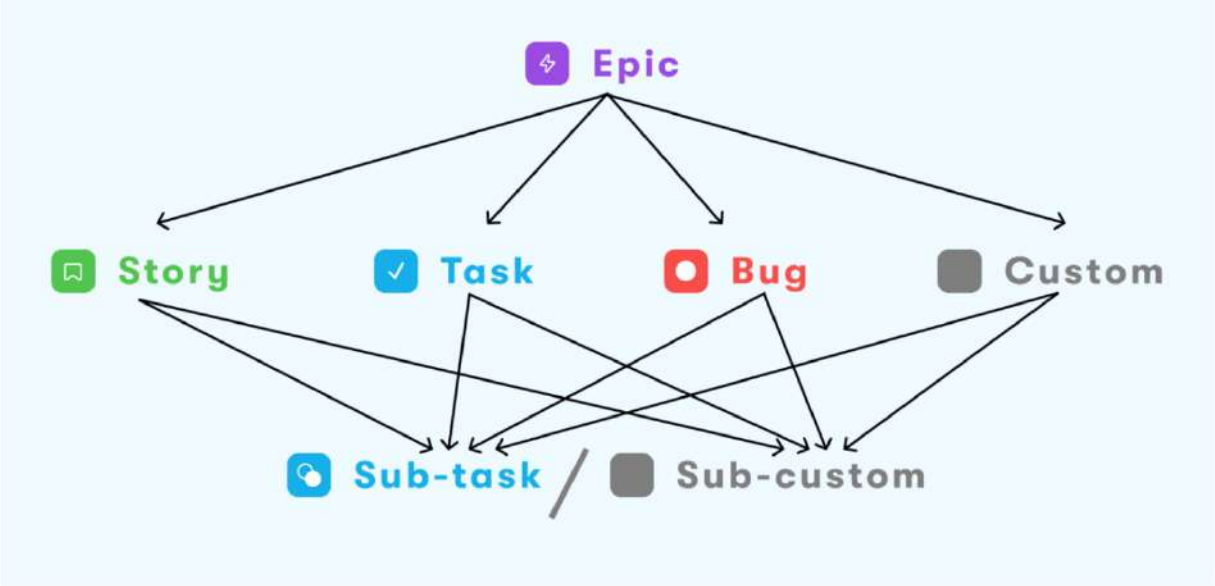


Рисунок 2.5 – Ієрархія в Jira

Також в Jira існує чудова можливість змінювати статуси завдань у довільному порядку. Використовуючи потрібні інструменти можна створити різні ланцюжки взаємодії між статусами завдань, додавати нові колонки із статусами до канбан дошки або ж робити перехід від одного статусу в інший (Рисунок 2.6)

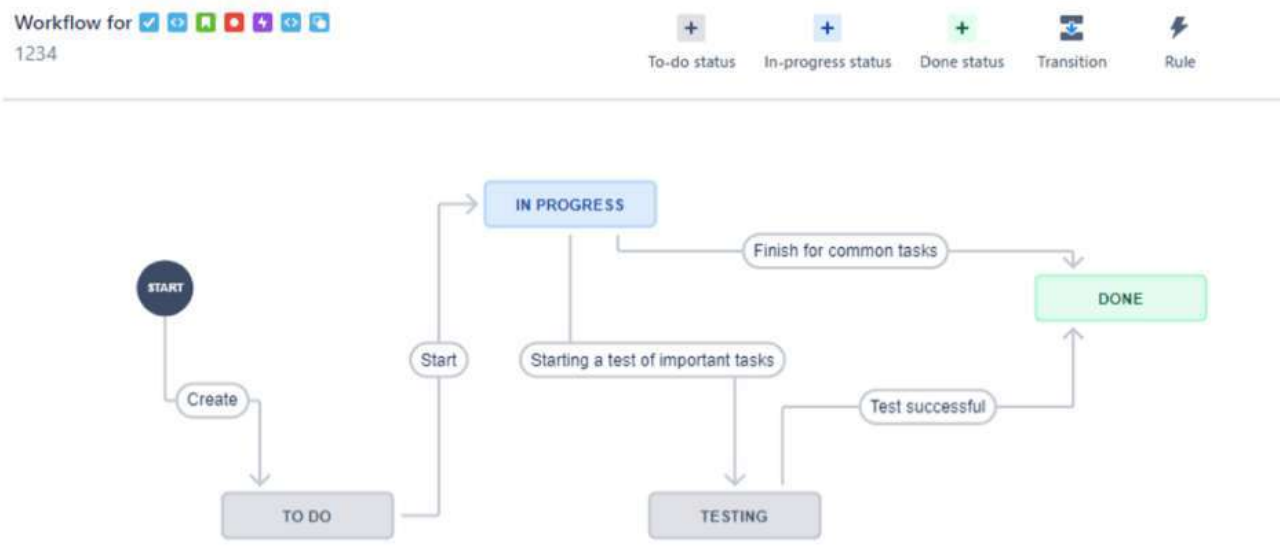


Рисунок 2.6 - Ланцюжок зміни статусу завдання в Jira

До основних переваг Jira можна віднести: Можливість налаштувати робочі процеси під свої потреби. З можливістю обрати різні шаблони та обрати використовувану методологію, Наявна можливість відстежувати та звітувати про поточний прогрес проекту, Можливість інтеграції з іншими зовнішніми інструментами та послугами, Повні можливості безпеки та контролю доступу завдяки розподіленням користувачів на ролі, Можливість відстеження затраченого часу та ресурсів на виконання завдань.

Також екосистема Jira має список недоліків: Вартість. Для малих команд Jira може бути дорогою. Оскільки потрібно платити абонентську плату, Складність. Jira може бути складною для використання. Особливо для користувачів, які не знайомі з програмним забезпечення для керування проектами, Обмежені звіти. Jira не пропонує багато опцій звітності, Jira не має своїх мобільних додатків для різних платформ, Відсутність підтримки для нестандартних інструментів та технік управління проектами.

Asana (Рисунок 2.7) – це безкоштовний на універсальний менеджер задач. Він допомагає слідкувати за процес створення та прогресом наявних завдань. Колекціонувати в одному місці всі необхідні завдання та файли для проекту.

Використовуючи Asana зручно розподіляти і ділити великі завдання на більш менші та зрозумілі завдання. З допомогою Asana можна з легкістю відслідковувати завантаженість вашого робочого дня та сортувати завдання у зручному для користувача порядку.

Використовуючи Asana є зручним як для великих команд із великою кількістю проектів та різного типу завдань, коли звичайні менеджери задач вже не можуть якісно виконувати адміністрування завдань. Хоч Asana і створювалась для команд, її можна використовувати як персональний записник, так само як і Notion. Всі базові та необхідні функції в Asana є.

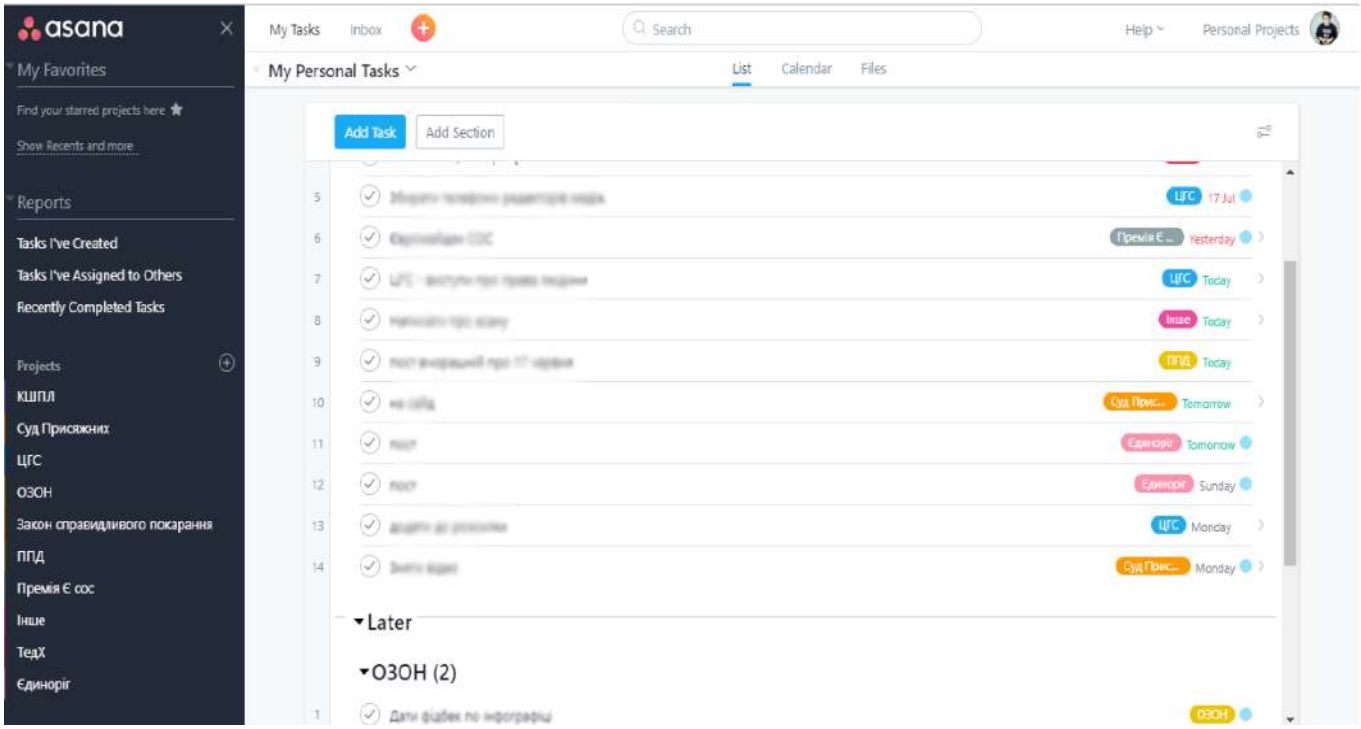


Рисунок 2.7 – Інтерфейс Asana

Для створених завдань можна побачити ким і коли воно було створено і які файли до нього були прикріплені чи написані коментарі. Усі члени команди можуть спостерігати історію змін у завдання, а отже жодна інформація не буде загублена.

У Asana є інтерфейс де користувач може бачити своє завантаження по днях, що є досить зручним наприклад коли потрібно призначити із кимось зустріч. Також можна синхронізувати свої заходи із Google Calendar. Цей функціонал є дуже зручним для менеджерів. Для менеджерів це можливість відслідковувати скупченість дедлайнів або спостерігати чи сповільнюється робота над проектом.

Також завдання можуть бути поділені на певні секції. Можна створювати проекти і вони будуть виглядати як канбан дошка як у Jira, Notion, Trello, а не звичайний список.

В Asana існує два типи акаунтів. Це Гість і Повний доступ. Для користувача із роллю Гість є можливість бачити лише певні проекти та завдання, але в той же час відсутня можливість бачити обговорення. Тип Гість зазвичай надають замовникам. Користувач в якого роль Повний доступ має можливість вносити

зміни у всі завдання та у всі файли. Зазвичай цю роль мають розробники проекту і в маленьких командах цю роль мають усі.

У Asana є ряд переваг: Збільшена продуктивність. Asana допомагає командам залишатися організованими і на правильному шляху, що дозволяє їм виконувати завдання швидше та ефективніше, Asana дозволяє командам легко працювати над проектами, ділитися ідеями та залишатися в синхронізації, Структуризована комунікація. Asana допомагає командам комунікувати ефективніше, надаючи інструмент для розмов та оновлень, Автоматизовані робочі процеси. Asana дозволяє командам автоматизувати повторювані завдання і процеси, заощаджуючи час та енергію людей, Asana надає детальні звіти та аналітику, щоб допомогти командам відстежувати прогрес і вимірювати успіх.

Також Asana має і недоліки: Asana не пропонує такої ж кількості інтеграцій як інші інструменти для управління проектами, Asana не пропонує такої ж кількості налаштувань як інші інструменти для управління проектами, Asana не пропонує такої ж кількості звітності як інші інструменти для управління проектами, Asana не пропонує такої ж кількості співпраці як інші інструменти для управління проектами, Обмежена підтримка користувачів. Asana не пропонує такої ж кількості підтримки користувачів як інші інструменти для управління проектами.

Провівши порівняльну характеристику аналогів таких як Notion, Asana, Jira на ринку можна зробити чіткі висновки:

1. Жодна із систем немає автоматизованого інструменту для того щоб можна було із впевненістю сказати що система зможе повністю на себе взяти зобов’язання менеджерів для розподілення та класифікації завдань. На даний момент це все потрібно виконувати у ручному режимі.
2. Спостерігається наявна проблема із інтеграцією із сторонніми сервісами.
3. Спостерігається наявна проблема із інтеграцією в приватні або навчальні сервіси.

4. Більшість із них, а саме Notion, Jira є платними сервісами. Але наявний функціонал із менеджментом задач покриває лише базові потреби та його не можливо автоматизувати в середині екосистеми проекту чи команди.
5. Для класифікації задач використовуються статуси задач, а не класифікаційні теги, що допомогло спростити роботу та навігацію по задачах.
6. Жодна із систем немає системи рейтингів команди розробників і не можливо відстежувати їхній прогрес.

2.2 Використання дерев ухвалення рішень для виконання класифікації задач.

Дерево ухвалення рішень – зазвичай використовують у галузях аналізу даних та галузі статистики. Дерево повинно містити такі частини як гілки(також називаються ребрами) та листя (Рисунок 2.8).

У гілках дерева пишуть параметри від яких залежить цільова функція, а в листя пишуть Sзначення для цільової функції. До інших вузлів дерева записують атрибути що діляться на випадки.

Для того щоб виконати класифікацію випадку, необхідно пройтись по всіх вузлах дерева до листка і повернути потрібне значення.

Такі дерева ухвалення рішень використовують під час інтелектуального аналізу даних, і їхня ціль базується на тому, щоб можна було створити таку модель, яка б виконувала прогноз значення для цільової функції або змінної використовуючи при цьому декілька змінних на вході.

Під час ходу руху по дереву рішень від кореня до листа, кожен лист є значенням цільової змінної. Внутрішні вузли відповідають за кожен із вхідних параметрів.

Також існує можливість вивчити дерево, шляхом поділення вихідних наборів на підмножини, які є заснованими на тестуванні значень атрибутів. Ці кроки

необхідно виконати повторно на кожній отриманні підмножині. Процес можна вважати завершеним коли значення цільової змінної дорівнюють або ж є такими самими в підмножині вузла, і в такий спосіб воно не надає цінності для прогнозів.

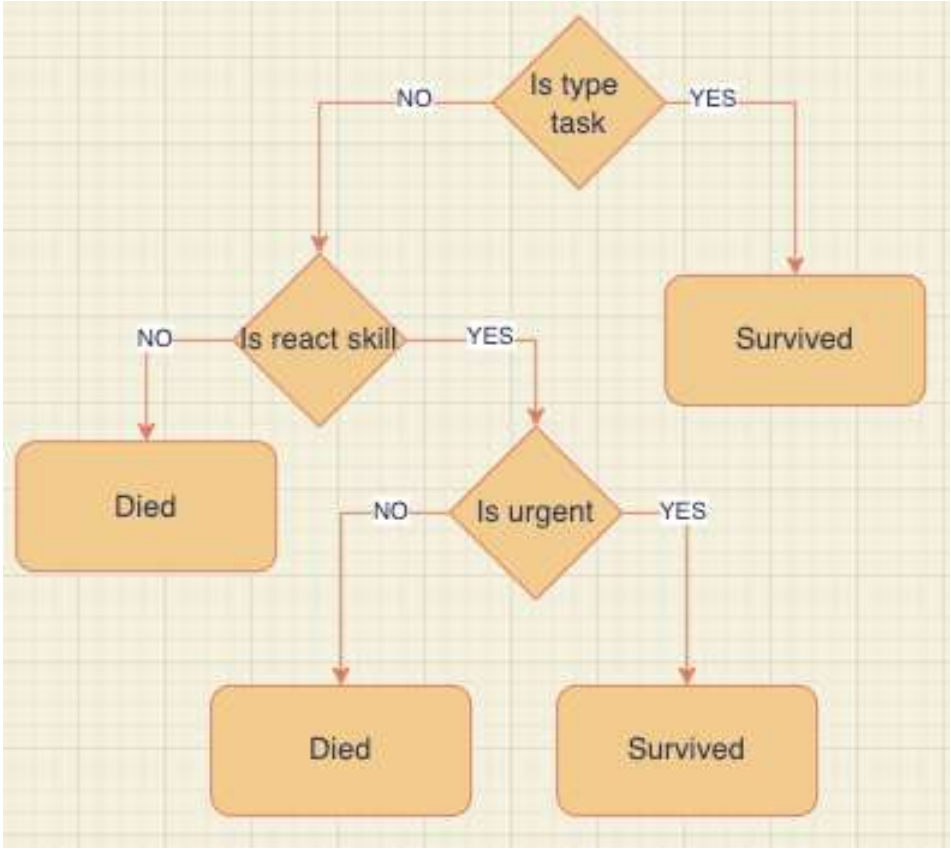


Рисунок 2.8 – Приклад дерева ухвалення рішень, дерева класифікації або регресивного дерева

На сьогоднішній день найбільш поширеною способом проходження дерева рішень є спосіб під час якого відбувається прохід від верху до низу або ще цей процес називають індукцією дерева рішень. Для інтелектуального аналізу даних, дерева рішень зазвичай використовують як обчислювальні або ж математичні методи, які в свою чергу допомагають класифікувати, описати, узагальнити вибірку даних, які описуються в наступний спосіб за формулою (2.1):

$$(x, Y) = \left(x_{1,} x_2, x_{3,} x_4 \dots x_k, Y\right), \quad (2.1)$$

де Y – залежна змінна цільової функції, вектор x - складається із вхідних змінних, які використовуються для виконання завдання.

Також цей прохід є гарним прикладом поглинаючого алгоритму. Також існують і інші способи для проходження дерева. Для підтримки ухвалень рішень в аналізі дерева рішень зазвичай використовують візуальний і аналітичний інструменти. Завдяки цим інструментам розраховуються значення конкуруючих альтернатив.

До складу дерева рішень входять три типи вузлів (зображені на рисунку 2.9):

1. Замикаючі вузли. Такі вузли зображають в вигляді трикутника на схемах дерев рішень.
2. Імовірнісні вузли. Такі вузли зображають у вигляді кола на схемах дерев рішень.
3. Вузли вирішення. Такі вузли зображають у вигляді квадрату на схемах дерев рішень.

Дерево не повинно мати в собі циклічні елементи. Це означає що кожен наступний лист пізніше може лише ділитися.

Існує проблема при побудові дерева рішень вручну. Зазвичай стикаються із проблемою розміру побудованого дерева рішень.

Тому для вирішення цієї проблеми використовують особливе системне програмне забезпечення. Дерева рішень зображують у вигляді символічної схеми. В такий спосіб його легше аналізувати і загалом сприймати.

Дерева рішень які використовуються для глибинного аналізу даних існують двох типів:

1. Регресійний аналіз дерева. Той випадок, коли результат який був прогнозованим, можливо розглядати як дійсне число.
2. Аналіз дерева класифікацій. Той випадок, коли клас є результатом.

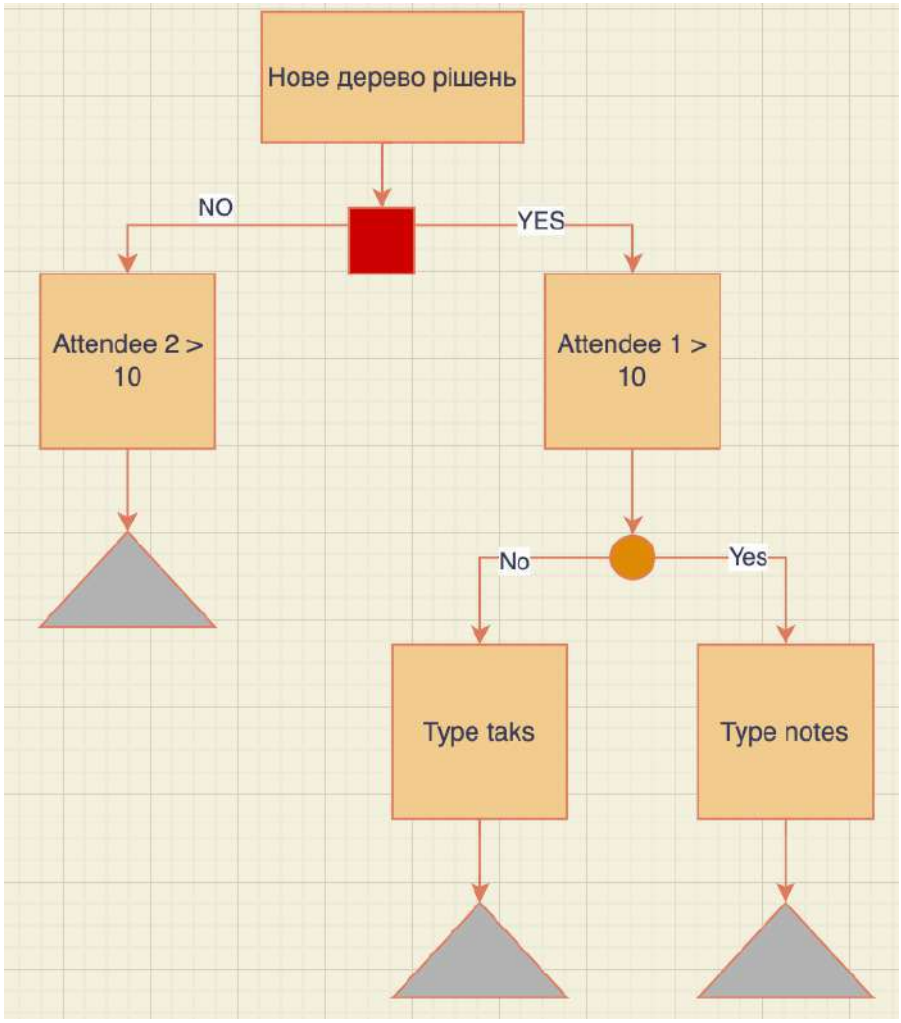


Рисунок 2.9 – Приклад зображення різних типів вузлів у дереві ухвалення рішень

Цих два аналізи дерева мають як і спільні риси так і відмінні риси. Наприклад до відмінних рис можна віднести підхід за допомогою якого можна визначати місце де необхідно виконати розбиття. Також існують методи використовуючи які можна створити більше ніж одне дерево прийняття рішень. Наприклад методи:

1. Обертання лісу. Це метод який використовує підхід де кожне наявне дерево рішень потрібно проаналізувати використовуючи метод головних компонент із використанням випадкових підмножин вхідних функцій.
2. Підвищенні дерева. Це метод, який використовується для класифікації типу проблем і для регресійного типу.

3. Класифікатор лісовий. Це метод, який використовується для рядів дерев рішень, для того щоб покращити ставки класифікації.
4. Мішок. Це один із перших методів для дерева рішень. Використовуючи даний метод можна побудувати одразу кілька дерев рішень. При чому не один раз із можливістю інтерполювати дані із заміною. А також створити дерева голосувань щоб виконати прогноз консенсусу.

Для того щоб побудувати дерево рішень із тестовими даними потрібно виконати такі кроки:

1. Потрібно обрати потрібний атрибут Q і потрібно добавити цей атрибут в корінь.
2. Потім для всіх можливих значень і потрібно виконати наступні дії. Залишити лише ті приклади де Q дорівнює і. Потім за допомогою рекурсії побудувати нове дерево в обраному нащадку.

Для того щоб обрати потрібний новий атрибут існує п’ять способів:

1. Використати алгоритм ID3
2. Використати алгоритм C 4.5.
3. Використати алгоритм CART.
4. Використати алгоритм CHAID.
5. Використати алгоритм MARS.

Алгоритм ID3 – це алгоритм який використовується для створення дерев рішень із попередньо підготовлених даних у сфері машинного навчання. Цей алгоритм також використовують у сфері обробки природньої мови. На першому етапі роботи алгоритму потрібно побудувати дерево із початковою множиною S у вузлі що знаходиться у коріння. Під час кожної ітерації, алгоритм проходить по всім не використаним атрибутам множини S і обчислює ентропію для обраних атрибутів. Наступним кроком алгоритм обирає атрибут з найменшим значенням ентропії. Далі виконується розбиття множини S за обраним атрибутом для отримання нових підмножин. Для кожної під множини із вибором атрибутів які не що до цього не були обрані раніше і виконується алгоритм рекурсивно. Для того щоб зупинити рекурсію необхідно виконати один із кроків:

1. Усі елементи в підмножині відносяться до одного класу. У такому випадку вузол маркується як клас прикладу і вузол перетворюється у листовий вузол.
2. Приклади не відносяться до одного класу і вже більше немає наявних атрибутів для проведення вибірки. В такий спосіб вузол позначається як найпоширеніший клас прикладів у підмножині і стає листовим вузлом.
3. У підмножині відсутні приклади. Таке може статися, коли у батьківській підмножині відсутній жодний приклад для того, щоб відповідати певному критерію обраного атрибуту. Далі має створитися листовий вузол, який необхідно позначити як найпоширеніший клас прикладів у множині батьківського вузла.

Алгоритм виконує побудову дерева в такий спосіб, що обраний атрибут представлять внутрішні вузли, де дані були поділені, а в підмножині вузла розгалуження представляють листові вузли.

Отже потрібно виконати чотири основних вимоги:

1. Обрахувати ентропію для кожного атрибуту I в наборі S.
2. Потрібно поділити множину S на менші підмножини де ентропія зведена до мінімуму, або ж приріст є максимальним.
3. Потрібно рекурсивним методом пройтись по підмножинах що залишились.

Алгоритм ID3 не може гарантувати оптимального рішення, цей алгоритм може лише сходитись до локального мінімуму, так як цей алгоритм використовує простий і прямолінійний евристичний підхід обираючи за допомогою підходу лише кращий локальний атрибут для того щоб виконати розбиття множини на під множини на кожній із ітерацій. Для того щоб покращити оптимальність цього алгоритму необхідно використовувати пошук з поверненням під час того як відбувається пошук оптимального дерева рішень. Але є також негативна сторона такого покращення це зменшення швидкості алгоритму.

Також для алгоритму ID3 доступне перенавчання. Для того щоб зменшити або уникнути процесу перенавчання потрібно обирати менші дерева, оскільки алгоритм ID3 не завжди повертає найменше дерево.

Алгоритм ID3 важко використовувати на даних які є неперервними. Якщо ж існують неперервні дані то пошук найкращого значення для розбиття буде забирати багато часу що не є оптимальним рішенням. Для того щоб підвищити точність алгоритму необхідно провести попередню обробку даних. Також у алгоритму ID3 існує можливість обчислювати інформаційний приріст замість ентропії. В такому випадку використовується для розбиття множини S атрибут із найбільшим інформаційним приростом.

Для розрахунку ентропії використаємо формулу 2.2.

$$\mathrm{H(S)} = \sum_{\mathrm{x}\epsilon\mathrm{X}} -\mathrm{p(x)}\ \log_2 \mathrm{p(x)}, \tag{2.2}$$

де S – набір даних, для якого розраховується ентропія, X – множина класів у S. p(x) – відношення числа елементів у класі x до числа елементів у множині S.

Для того щоб порахувати інформаційний приріст використаємо формулу 2.3

$$IG(S,A) = H(S) - \sum_{t\in T} p(t)H(t) = H(S) - H(S|A), \tag{2.3}$$

де T – підмножини, створені після розбиття множини S за атрибутом A. H(S) – ентропія множини S. H(t) – ентропія підмножини t, p(t) – відношення числа елементів у t до числа елементів у множині S.

C4.5 алгоритм використовується для класифікації. Алгоритм є статистичним класифікатором. Він є покращений варіантом алгоритму ID3. Використовуючи інформаційну ентропію алгоритм будує дерева рішень в такий самий спосіб як і ID3. Де дані S це набір класифікованих зразків, де кожен зразок містить у собі вектор n-го розміру і де кожен елемент повинен містити значення атрибуту або ознаки, клас тощо. Перебираючи кожен етап цей алгоритм виконує набір правил на кожному вузлі. Він обирає атрибут даних де розбиває набір зразків на підмножини

керуючись також ознаками що належать до одного класу. Різниця ентропії слугує як критерій розбиття. Атрибут який містить найбільший інформаційний приріст обирається для того щоб прийняти рішення. Також як і ID3 алгоритм рекурсивним методом повторює дії на розбитих підмножинах на попередньому етапі. Також як і ID3 алгоритм має бути умова для виходу із рекурсії. Можливо три сценарії випадків які зустрічаються:

1. Випадок коли з'являється екземпляр класу який до цього раніше не був оброблений. Створюється вузол у дереві на один рівень вище використовуючи при цьому математичне очікування.
2. Випадок коли ознаки які є не забезпечує потрібного інформаційного приросту. В такому випадку створюється вузол у дереві на один рівень вище використовуючи при цьому математичне очікування.
3. У випадку коли всі дані належать до одного класу. В такому випадку необхідно створити новий вузол, який буде казати обрати саме цей клас.

Тобто для того, щоб виконати алгоритм потрібно виконати усі описані кроки:

1. Потрібно виконати перевірку на наявність описаних головних випадків.
2. Потім для кожного наявного атрибута знайти коефіцієнт інформаційного приросту.
3. Потім необхідно створити новий вузол який розіб'є множину використовуючи коефіцієнт інформаційного приросту.
4. Використовуючи рекурсивний метод потворити вище описані дії на створених раніше підмножинах та створити вузли у якості листя.

Завдяки алгоритму C4.5 було покращені такі критерії алгоритму ID3:

1. Можливість виконувати обробку атрибутів із різними витратами.
2. Можливість виконувати обрізання дерев. Цей метод дозволяє повернутися назад і видалити гілки які не допомагають приймати правильні рішення.

3. Можливість обробки навчальних даних навіть із відсутніми значеннями атрибутів.
4. Також появилась можливість обробляти як дискретні так і неперервні дані

Також алгоритм C4.5 має покращену версію яка C5.0 а також ще один алгоритм See5.

Алгоритм CART призначений для створення бінарних дерев рішень, де кожен вузол має лише два нащадки. При використанні алгоритму CART поведінка об'єктів виділеної групи визначається як відсоток модального значення вихідної ознаки. На кожному кроці побудови дерева правило, що формується у вузлі, ділить задану кількість прикладів на дві частини - частину, де правило виконується (right) і частину, де правило не виконується (left).

Алгоритм CART забезпечує можливість виявлення детермінацій, якщо вони існують у досліджуваній сукупності, а також дозволяє пошук всіх можливих значень вихідної ознаки, що мають відповідний вираз. Цей метод може бути застосований для номінальних та порядкових передбачуваних змінних. CART метод використовується для прогнозування номінальних та порядкових змінних. При цьому перебираються всі можливі варіанти розгалуження для кожного вузла, а найкращий показник отримується за допомогою оцінки функції передбачуваної змінної. Для номінальної передбачуваної змінної, що має k різних значень, є рівно $2(k-1) - 1$ варіантів розбиття множини її значень на дві частини. Для порядкового передбачення, що має k рівнів, є k-1 точок розгалуження. При побудові дерева рішень для багатьох передбачень з багатьма рівнями значень необхідно переглянути велику кількість варіантів розгалуження. Для оцінки якості побудованої моделі потрібен індикатор. Алгоритм CART використовує оціночну функцію, що базується на інтуїтивному понятті зменшення невизначеності в вузлі. Наприклад, якщо вузол містить по 50 прикладів двох класів, то максимальна невизначеність буде знаходитися в ньому. Знаходження розбиття, яке розділить дані на дві підгрупи 40:5 та 10:45, призведе до зменшення неоднорідності, а знаходження розбиття, яке створить підгрупи 50:0 та 0:50, призведе до повного

зникнення невизначеності. Індекс Gini використовується для формалізації цієї ідеї для набору даних, що містить n класів. У підсумку, найкращим буде те розбиття, для якого максимальна величина. Таким чином, при побудові «дерева рішень» методом CART виконується пошук такого варіанту розгалуження, при якому максимально зменшується значення показника Ginisplit(T).

Для того щоб порахувати оцінку якості моделі використаємо формулу 2.4

$$Gini(T) = 1 - \sum_{i-1}^{n} p_i^2, \tag{2.4}$$

де p – ймовірність класу i в T .

Також є механізм відсікання. Цим механізмом, що має назву мінімальна вартість-складність відсікання дерева алгоритм CART принципово відрізняється від інших алгоритмів побудови дерев рішень. У розглянутому алгоритмі відсікання - це компроміс між отриманням дерева «відповідного розміру» і отриманням найбільш точної оцінки класифікації. Відсікання важливо не тільки для спрощення дерева, але і для уникнення перенавчання дерева рішень. Метод полягає в отриманні послідовності дерев, що зменшуються, але дерева розглядаються не всі, а тільки найкращі представники.

Також існує механізм перехресної перевірки. Це є спосіб вибору остаточного дерева, коли набір даних має невеликий об'єм або записи набору даних так специфічні, що розділити набір на навчальну та тестову вибірки неможливо.

Переваги методу CART:

1. Цей метод є непараметричним, тобто для його застосування не потрібно розраховувати різні параметри імовірнісного розподілу.
2. Для застосування алгоритму CART не потрібно заздалегідь вибирати змінні, які будуть брати участь у аналізі: змінні вибираються безпосередньо під час аналізу на підставі значення індексу Gini.
3. Алгоритм CART легко справляється із викидами: механізм розбиття, який включений в алгоритм, просто поміщає викиди в окремий вузол, що дозволяє очистити наявні дані від шуму.

4. Перевагою є також швидкість алгоритму.
5. Для того щоб використати алгоритм не потрібно виконувати жодних припущень чи виконувати припущення перед тим як провести аналіз.

Недоліки методу CART:

1. Дерева рішень не завжди є стійкими.
2. Не підходить для побудови дерев із складною структурою. Алгоритм CART може не правильно визначити структуру даних.

Також потрібно вміти регулювати глибину дерева. Завдяки цій техніці можна зменшувати розмір дерева. Проблема яку необхідно вирішити це проблема оптимального розміру дерева. Дуже складно зробити прогноз що під час масштабування дерева результат дозволить зменшити помилку. Це ще називається як ефект горизонту. Але попри все загальний метод полягає в тому щоб видаляти зайві вузли, якщо їх результатом можна знехтувати. Тут дуже важливо не вирізати розгалуження які впливають на кінцевий результат. Також можна використати вище описану перехресну перевірку. Існує багато методів, але відрізняються вони лише одиницями вимірювання продуктивності. Зменшення дерева можливо або знизу і вверх або ж зверху і вниз. Якщо починати із знизу і рухатися до верху в такий спосіб скорочується число листя дерева, а якщо рухатися зверху і до низу то процес обрізки починається із кореня. Найпростіший метод – зменшення помилки дерева. Проходячись по всім листя і підміняючи значення на найпопулярніший клас. Якщо ж у випадку підміни результат не змінився то заміна відбувається.

Також для класифікації можна використати метод випадкового лісу. Випадковий ліс також використовують у сфері машинного навчання для регресії, класифікації тощо. Цей метод будує багато дерев рішень, і під час навчання моделі знаходить значення випадкової величини, що трапляється найчастіше для наявних класифікацій. Або ж знаходить значення для усередненого прогнозу згенерованих дерев.

Метод випадковий ліс має ряд переваг:

1. Можливість працювати в багатьох потоках одночасно.

2. Існує можливість для внутрішньої оцінки моделі здатності до узагальнення.
3. Наявність окремих підходів для оцінювання значущості для окремих ознак.
4. Можливість обробки як дискретних ознак так і безперервних. Також існують методи для того щоб можна було побудувати дерева за наявними даними з пропущеними ознаками.
5. Метод не чутливий до масштабування.
6. Можливість обробляти дані з великим об’ємом класів і ознак.

Метод також має декілька недоліків: Метод схильний до перенавчання, Занадто великий розмір моделей які отримуються після виконання алгоритму. Потрібна велика кількість пам’яті для зберігання таких моделей.

Для виконання алгоритму потрібно зробити наступні кроки:

1. На першому етапі необхідно згенерувати випадкову вибірку із основної вибірки із розміром n. В такий спосіб у нову вибірку деякі елементи можуть потрапити декілька разів а деякі взагалі ні.
2. Потім необхідно обрати m кількість ознак зі M.
3. Наступним кроком необхідно побудувати дерево рішень, де під час створення нового вузда потрібно обрати ознаку яка входить до складу m раніше обраних ознак які були обрані випадковим чином. Де вибір найкращого можливий наприклад використовуючи критерій Джині, або ж можна використати критерій приросту інформації.
4. Потрібно розділити ознаку на два класи.
5. Використовуючи критерій Джині або ж критерій приросту інформації потрібно виміряти гомогенність у двох класах.
6. Потрібно знайти значення для розділення множини по ознаці де досягнутий максимальний гомогенний клас.
7. Потрібно побудувати дерево до того моменту коли закінчиться вибірка. Не можна на цьому етапі робити відсікання.
8. Рекурсивним методом повторити вище вказані кроки.

2.3 Методи для розподілення задач

Для розподілення задач можна обрати два підходи. Перший це розв’язання за допомогою лінійної оптимізації або ж задач лінійного програмування. Або ж із використанням генетичного алгоритму.

Лінійне програмування це – метод за допомогою якого можна досягти найкращого виходу математичної моделі. Вимоги у математичні моделі мають бути подані через лінійні відношення. Також вважають лінійне програмування особливим випадком для математичної оптимізації. Іншими словами лінійне програмування є методом для оптимізації лінійної цільової функції, яка має обмеження у вигляді лінійної нерівності і лінійних рівнянь. Задачі максимізації функції по факту зводяться до того щоб виконати задачу мінімізації функції шляхом зміни знаки усіх коефіцієнтів на протилежні знаки. Існує декілька методів розв’язання: Двоїстий симлекс-метод, Симлекс метод, Метод потенціалів.

Двоїстий симплекс метод був розроблений після симплекс методу. Цей метод по факту є методом для розв’язання задачі яка дає нижню межу задачі під час мінімізації але яка в той же час може давати різні розв’язки для прямої та двоїстої задачі лінійного програмування. Метод був описаний в термінах для вихідної задачі.

Симплекс метод це метод який для випадку задачі лінійного програмування є узагальненням методу потенціалів. Був розроблений у 1949 році. Нехай задачу лінійного програмування в канонічному вигляді представлено у формулі 2.5

$$\sum_{j=1}^{n} c_j x_j \rightarrow max, \tag{2.5}$$

де x – вектор змінних, c – заданий вектор .

Метод потенціалів це метод також використовується для вирішення так званої транспортної задачі, де потрібно виконати пошук оптимального плану перевезення вантажу із пункту А до пункту Б. Був розроблений ще в 1940 році радянськими вченими.

Використовуючи лінійне програмування можна виконати моделювання різних проблем у плануванні. Використовуються також для вирішення задач перевезення, телекомунікацій тощо. Також у задачах призначення та маршрутизації. Галузі використання є економіка, бізнес.

Генетичний алгоритм це алгоритм який також використовується для вирішення різного роду задач оптимізації та моделювання використовуючи при цьому підходи що є схожими до біологічної еволюції. Метод використовує еволюційний алгоритм пошуку використовуючи послідовний відбір, варіації та комбінування параметрів які необхідно знайти. Особливу увагу у генетичному алгоритмі приділяють використанню оператора схрещення. Оператор схрещення виконує операцію яка називається ре комбінуванням для рішень кандидатів. Таку ж у роль схрещення як і у природі.

Для того щоб реалізувати алгоритм необхідно закодувати задачу в такий спосіб, щоб вирішення можна було представити у вигляді масиву. Цей масив має бути схожий до того як складається хромосома. Використовуючи випадковий чинник потрібно створити першу популяцію елементів. Потім необхідно провести оцінку кожного такого елемента використовуючи оцінку допасованості, ця оцінка впливає на можливість виживання елемента. Наступним кроком обираються особи які можна схрестити. До обрано елемента використовують оператор мутації та схрещення. І в такий спосіб буде створено нове покоління. Потім потрібно повторити вище описані кроки до моменту коли буде виконано критерій для зупинки алгоритму. Цим критерієм може бути:

1. Пошук над оптимального або глобального рішення.
2. Перевиконання допустимого числу мутації і створення поколінь, які були виділення для еволюції.
3. Перевиконання допустимого часу для мутації і створення поколінь, які були виділення для еволюції.

Генетичний алгоритм використовується в пошуку важких та великих просторах пошуку.

Існує сім базових кроків для виконання алгоритму:

1. Необхідно створити початкову популяцію елементів.
2. Провести для кожного елементу популяції оцінку допасованості.
3. Далі виконувати циклічно кроки 4,5,6,7,8 до моменту коли вичерпається заданий час або ж ліміт на кількість популяцій для закінчення алгоритму або буде знайдений найоптимальніше вирішення задачі.
4. Виконати селекцію із поточної популяції.
5. Виконати мутацію елементів або ж схрещення.
6. Виконати оцінку допасованості для кожного елементу популяції.
7. Створити нове покоління.

Генетичний алгоритм використовується в пошуку важких та великих просторах пошуку.

Виконуючи перший крок потрібно використовуючи випадковість створити першу популяцію. У випадку якщо популяцію буде виглядати геть не життєздатною, алгоритм в швидкому темпі перетворить популяцію на життєздатну.

На першому етапі достатньо зробити популяцію із елементів на яких можна виконати оцінку допасованості. Після першого етапу у нас буде популяція H, яка буде містити в собі N елементів.

На наступному етапі відбору відбувається вибір елементів які потраплять в наступну популяцію із вже зміненими параметрами.

Існує декілька методів для виконання такого відбору. Зазвичай ймовірність виживання особи залежить від значення дропасованості. Кількість відібраних елементів є параметром алгоритму і він визначається ще перед першим етапом. Після перевірки створюється нова популяція, а елементи які не були відібрані просто гинуть.

На етапі розмноження потрібно два елементи для того щоб створити нового нащадка. Головна вимога щоб нащадок міг успадкувати гени обох батьків, перемішавши їх якимось чином. Існує два варіанти такого змішання це кросинговер і рекомбінація.

У генетичному алгоритмі присутня проблема коли є недостача різноманітності в особах популяції. Для того щоб цього уникнути обирають підхід коли відбір йде не з уже відфільтрованих елементів а з усіх в популяції. Для мутацій необхідно обрати деякі елементи із популяції і змінити їх за зарання заданими параметрами.

2.4 Висновки

В результаті проведеного аналізу було виявлено, що існуючі підходи до системи розподілення та класифікації задач мають такі як свої переваги та недоліки.

Було проаналізовано багато методів та алгоритмів для вирішення різного класу задач. Для виконання задачі класифікації було розглянуто методи дерева прийнятті рішень та випадковий ліс. Про аналізувавши ці методи більше детально, дослідивши різні наявність різних внутрішніх реалізацій самих алгоритмів можна сказати, що для того щоб додавати класифікаційні теги до завдання краще підходить алгоритм дерева прийняття рішень.

Для такого дерева також потрібно буде згенерувати вибірку із уже наявних завдань та створити допоміжний алгоритм який допоможе зберігати а трансформувати ці дані для виконання класифікації. Для системи розподілення задач було досліджено підхід із використанням лінійного програмування та генетичний алгоритм. Проаналізувавши метод лінійного програмування дійшов до висновку що для того щоб знайти найкращого кандидата потрібно створити чітку математичну модель і правильно скласти цільову функцію для пошуку кандидата.

Генетичний алгоритм буде більш ефективний якщо правильно використовувати процес мутації та схрещування. Оскільки генетичний алгоритм потребує зарання визначеної кількості етапів еволюції або знати обмеження по часу.

Було обрано як метод лінійне програмування. Але перед тим як виконувати пошук використовуючи метод лінійного програмування. Потрібно створити також дані для використання пошуку, а також метод трансформації цих даних.

Також потрібно створити підхід де можна буде використати систему рейтингів із учасників які можуть впливати на кінцевий результат і цільову функцію, оскільки має гнучкість параметрів у цільові функції.

А аналіз уже наявних рішень на ринку показав що системи не мають жодного інструменту для того щоб можна було здійснити автоматичну класифікацію задач або ж виконати розподілення задач між виконавцями. В Notion є віртуальний помічник Notion AI. Він має схожі можливості до Chat GTP 3 версії. Але його задача полягає в підтримці користувача і допомозі в різному роді креативних задач.

Але у всіх цих системах досі існує можливість моніторингу задач менеджером без будь якої підтримки автоматизації розподілення. Отже розроблена система повинна:

1. Використовуючи підхід із використанням лінійного програмування розробити алгоритм і цільову функцію для пошуку найкращого виконавця завдання.
2. Використовуючи підхід із використанням дерев ухвалення рішень розробити алгоритм і функцію для генерування вхідних даних для можливості виконати класифікації завдання та присвоїти задачі відповідні класифікаційні теги.
3. Розроблена система розподілення та класифікації задач має бути легко інтегрована у робочий процес.
4. Потрібно розробити алгоритм для загального процесу який зможе на основі вхідних даних виконати за один прохід як класифікацію так і розподілення за найменш можливий проміжок часу.
5. Розробити блок схеми алгоритмів та зробити графіки для відстежування коректності виконання поставлених задач.

3 АЛГОРИТМИ ТА ТЕХНОЛОГІЯ ДЛЯ ВИРІШЕННЯ ЗАДАЧ

3.1 Алгоритм вирішення оцінювання та класифікації задач

Для того щоб приступити до етапу планування та розробки алгоритму який буде вирішувати задачу оцінювання та класифікацію задачі, як уже було описано в попередніх розділах необхідно створити відповідний дата сет на основі якого буде будуватися алгоритм.

На етапі коли потрібно згенерувати дата сет потрібно в першу чергу визначити а у який спосіб буде генеруватися потрібний нам дата сет і яку структуру даних він повинен мати.

Для цього нам необхідно визначити перелік як вхідних даних так і перелік параметрів які нам необхідні для того щоб виконати функцію яка буде генерувати нам потрібну структуру даних, яку ми буде зберігати в базі даних.

Для того щоб скласти перелік вхідних даних та параметрів нам також необхідно перед цим визначитись в який саме спосіб і оперуючи якими даними ми будемо оцінювати наших постійних користувачів.

Під час процесу виконання поставлених задач користувач здобуває певний перелік навичок. Або також може закріпити отримані знання. А також поновити актуальність своїх знань. В такий спосіб ми отримали першу потрібну нашу сутність. Це навичка. Зазвичай навички поділяються на дві групи це hard або ж “Жорсткі” і soft або ж м’які навички.

Hard skills це спеціалізовані знання та професійні навички яким можна навчити або вивчити самому а також ці навички можна виміряти та протестувати, бо це також є тими практичними навичками, які ми отримуємо з часом і з досвідом. Тобто hard skills впливають на пряму можливість кандидата виконувати поставлене завдання. Отже для hard skills є характерними такі риси:

1. Наявний перелік знань та умінь, які можна об’єктивно оцінити та провести перевірку.
2. Наявність різних документів про здобуття навичок. Також це можуть бути документи про освіту.

3. Сертифікати яка вказують на рівень отриманих або володіння тих чи інших знань.

Згідно із дослідженням в LinkedIn, до найбільш популярних hard skills в IT сфері належать:

1. Data Science.
2. Video production.
3. Sales management.
4. Affiliate Marketing.
5. Scientific computing.
6. Business Analysis.
7. UX Design.
8. AI. Artificial Intelligence.
9. Cloud Computing.
10. Blockchain.

Soft skills це універсальні компетенції, ці навички зазвичай набагато важче виміряти. Вони допомагають досягти успіху і тих чи інших галузях та ролях. Також іноді їх називають особистими якостями, так як вони залежать від характеру людини і накопичуються із досвідом. Найчастіше ступінь прояву залежить від особливостей характеру, від типу особистості, темпераменту тощо. Серед найбільш важливих і відомих soft skills відносять:

1. Ділова хватка. Мається на увазі процес розуміння, чого потребує бізнес, на якій стадії знаходиться та в якому напрямку необхідно його розвивати.
2. Допитливість. Внутрішнє відчуття до пошуку на важливі та актуальні питання. Зазвичай це спонукає задавати питання ”Чому?”, бо як відомо однієї відповіді не є достатньо.
3. Емоційний інтелект. Ця навичка дозволяє розуміти почуття та емоції оточуючих людей і свої власні. Вміння керувати своїми власними емоціями та мати можливість впливати на емоції інших людей.

4. Здатність розв’язувати поставлені задачі. Вміти ефективно визначати наявні проблеми та знаходити ресурси та шляхи для того щоб їх вирішити.
5. Можливість критично мислити. Це здатність базуючись на наявні інформації усвідомлювати проблему, виконувати аналіз та знаходити і приймати найоптимальніше рішення проблеми.
6. Мати ефективну комунікацію. Ця навичка дозволяє чітко, конкретно, та ясно висловлювати своє бачення ситуації та власні думки, при цьому враховуючи технічну складову аудиторії. Тобто для технічних людей та тих які не працюють у даній сфері.

Для IT ринку найчастішими і найпотрібнішими soft skills є:

1. Вміти працювати в команді.
2. Мати критичне мислення.
3. Лідерство.
4. Креативність.
5. Вміти вкладатися у дедлайн.
6. Відповідальність.
7. Дисциплінованість.

Отже у підсумках головні відмінності між hard і soft skills:

1. Для того щоб розвивати hard skills необхідний інтелектуальна складова. Для розвитку Soft skills потрібна, емоційна складова.
2. Зазвичай вимоги до hard skills залишаються не змінними незалежно від компанії і корпоративної культури. Наприклад, правила створення програмного коду будуть однакові для будь якого розробника. Soft skills у свою чергу є більш мінливими та ситуативними. Правила ведення ефективного мовлення будуть залежати від аудиторії, до якої звертаються.
3. Оволодіти hard skills можна, наприклад, в різних навчальних закладах, курсах тощо. Зазвичай ці навички можна поділити на відповідні рівні складності. І для того щоб отримати новий рівень потрібно успішно

скласти екзамен з опанування потрібних навичок. На відміну від hard skills для soft skills не існує покрокового плану. Зазвичай людина або має потрібні навички з народження або ж отримує їх із досвідом шляхом проб та помилок.

4. Soft skills зазвичай набагато довше і складніше опанувати ніж hard skills.

Отже визначившись із типами навичок які бувають нам необхідно на даному етапі скласти свій перелік базових навичок якими повинен володіти користувач для того щоб можна було класифікувати задачу враховуючи необхідні навички які потрібні для виконання наявної задачі. Перерахуємо навички які будуть використовуватись у нашій системі:

1. Рівень англійської. (English skill).
2. Вміння використовувати час для самого себе. (Self management).
3. Вміння керувати командою. (Team management).
4. Досвід у галузі. (Experience duration).
5. Спокій користувача. (Calm).
6. Стабільність. (Stability).
7. Відповідальність. (Responsibility).
8. Якість. (Quality).
9. Дизайн навички. (Design skills).
10. Інтерфейс користувача. (UI skills).
11. Навички сервера. (Backend skill).
12. Навички розгортання. (Devops skill).
13. Навички із мобільної платформи. (React Native skill).
14. Навички тестування. (Testing skill).
15. Швидкість опанування нового матеріалу. (Learning speed).
16. Швидкість написання коду. (Development speed).

Сформований перелік навичок у схемі виглядає так як на рисунку 3.1

Рисунок 3.1 – Перелік базових навичок користувача які необхідні для виконання оцінки користувача та також для класифікації задачі

Оскільки кількість базових навичок дорівнює шістнадцяти, то можна із впевненістю сказати що для такої системи максимальна кількість всіх оцінок буде становити сто шістдесят балів. Для того щоб дізнатися в якому процентному співвідношенні має кожна навичка – розділимо кожну навичку на одну цілу і шість десятих. І в такий спосіб ми отримаємо шість цілих і двадцять п’ять сотих відсотка

припадає на кожну навичку. Це максимальне значення яке може отримати кожна навичка. Також для формул введемо нові додаткові позначення на рисунку 3.1 де навички які вже розділення на задану величину.

На наступному етапі потрібно виділити чотири основних підгрупи для вище перерахованих навичок. Ці підгрупи включать в себе:

1. Навичка. Skill (S).
2. Менеджмент група. Management (M).
3. Група надійності. Reliability (R).
4. Група перспектив. Prospects (P).

Далі потрібно порахувати максимальну оцінку в процентному співвідношенні для кожної ново створеної підгрупи. Для того щоб порахувати суму всіх навичок які відносяться до групи Skill використаємо формулу 3.1.

$$S = \sum(DES(m), UIS(m), BS(m), DOS(m), RNS(m), TS(m)), \quad (3.1)$$

де m – це максимальна оцінка в процентному співвідношенні.

Наступним кроком потрібно порахувати середнє та відносне значення кожного навичку для цієї групи. Для того щоб порахувати середнє потрібно просто додати всі навички і поділити на їх кількість. Для того щоб порахувати середнє відносне потрібно знайти спочатку коефіцієнт, а для цього суму всіх навичок помножити на їх кількість і розділити на сто. Отримаємо 0,375 А потім отримане суму розділити на отримане значення.

Далі необхідно порахувати потрібні значення для решти підгруп Management(M), Reliability (R), Prospects(P). Для того щоб порахувати суму всіх навичок які відносяться до групи Management використаємо формулу 3.2.

$$M = \sum\{E(m), SM(m), TM(m), ED(m), S_a\}, \quad (3.2)$$

де S_a – це раніше обрахована величина, тобто середнє значення для Management, а m – це максимальна оцінка в процентному співвідношенні.

Далі знову потрібно порахувати середнє та відносне значення кожного навичку для цієї групи. Діємо за тим самим алгоритмом. Коефіцієнт дорівнює 31,25. Для того щоб порахувати середнє потрібно просто додати всі навички і поділити на їх кількість. Для того щоб порахувати суму всіх навичок які відносяться до групи Reliability використаємо формулу 3.3.

$$R = \sum\{Ca(m), SM(m), TM(m), ED(m), Re(m)\}, \quad (3.3)$$

де m – це максимальна оцінка в процентному співвідношенні.

Знову рахуємо середнє та відносне значення кожного навичку для цієї групи. Коефіцієнт дорівнює 0,25.

Для того щоб порахувати суму всіх навичок які відносяться до групи Prospects використаємо формулу 3.4.

$$D = \sum\{LS(m), DS(m), S\}, \quad (3.4)$$

де S – це раніше обрахована величина, тобто середнє значення для Prospects, а m – це максимальна оцінка в процентному співвідношенні.

Знову рахуємо середнє та відносне значення кожного навичку для цієї групи. Але для обрахунку відносно значення використаємо формулу 3.5.

$$D(r) = \frac{(S_{max} \cdot 2 + S(r))}{100}, \quad (3.5)$$

де S – це сума всіх навичок які відносяться до групи Skills, а S(r) це відносне значення групи Skills яке ми отримали у формулі 3.3.

Наступним кроком введемо такі значення як Mandatory tags. Тобто це ті групи тегів які обов’язково повинні будуть визначені для описаного завдання. Обов’язкових тег груп буде п’ять. Також потрібно передбачити у системі можливість створення додаткових групи тегів які не повинні впливати на п’ять

базових і зарання визначених. Кожна тег група яка зарання визначена або ж створення в процесі використання інтегрованої системи може містити в собі підмножину величиною N.

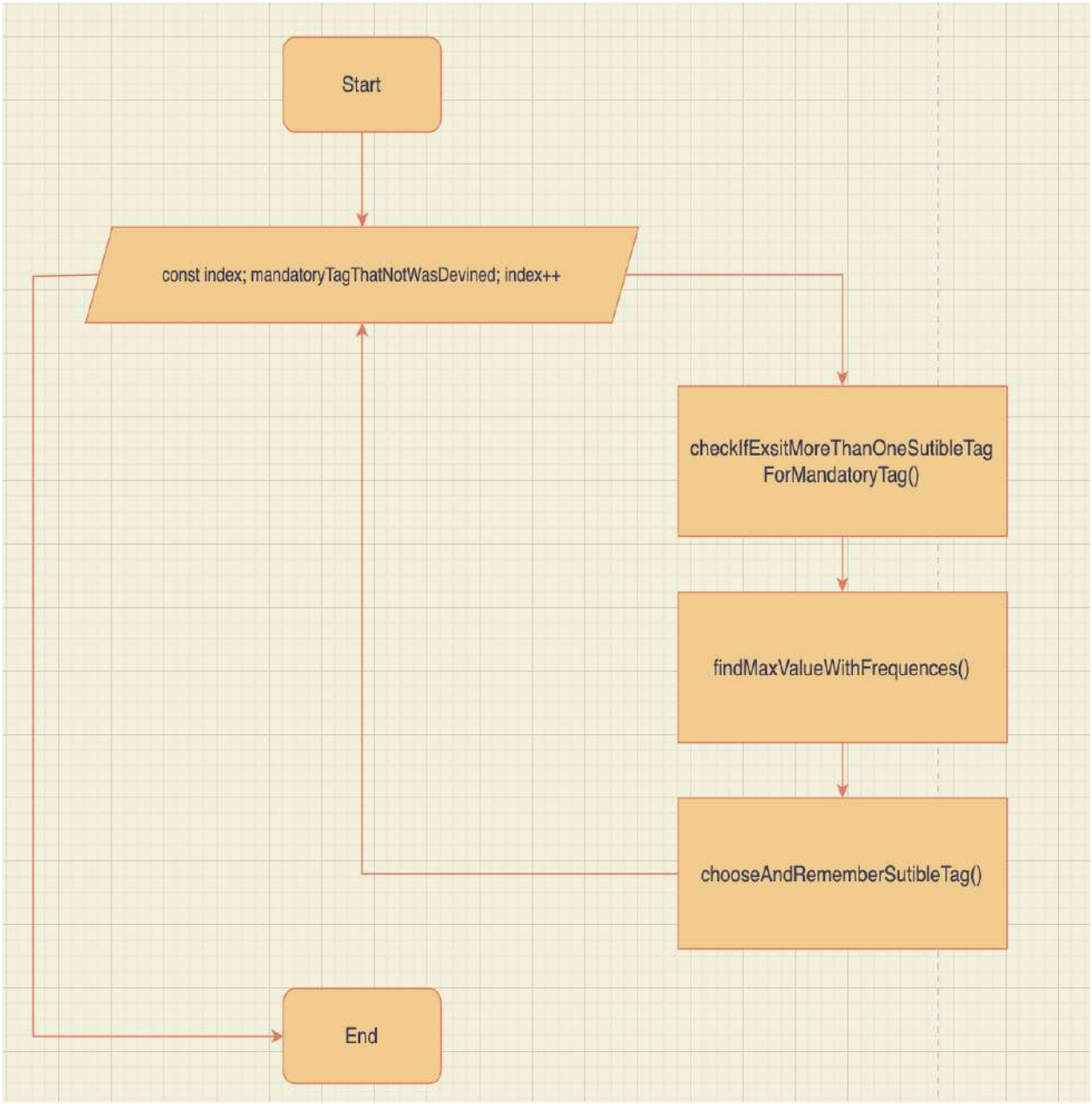


Рисунок 3.2 – Алгоритм який реалізовує лише пошук найкращого елементу із запропонованих. Тобто це реалізація формули 3.6

П’ять основних груп будуть:

1. Пріоритети. (Priorities).
2. Проекти. (Projects).

3. Необхідні навички. (Skills).
4. Тип. (Types).
5. Фаза виконання задачі. (Phase).

Для визначення тегів які відносяться до задачі будемо в основі використовувати формулу 3.6.

$$Tags(N) = \sum_{i=1}^{N} \left(\underset{max}{\longrightarrow} (Priorities(SubPriorities(i))) \right) + \sum_{i=1}^{N} \left(\underset{max}{\longrightarrow} \left(Projects\left(SubProjects(i)\right) \right) \right) + \sum_{i=1}^{N} \left(\underset{max}{\longrightarrow} (Phases(SubPhases(i))) \right) + \sum_{i=1}^{N} \left(\underset{max}{\longrightarrow} (Skills(SubSkills(i))) \right) + \sum_{i=1}^{N} \left(\underset{max}{\longrightarrow} (Types(SubTypes(i))) \right), \quad (3.6)$$

де N – це натуральне число, а SubPriorities ∈ Priorities, Sub Projects ∈ Projects, SubPhases ∈ Phases, SubSkills ∈ Skills.

Для реалізації розрахунку формули буде створена функція і вона буде називатися choose most used tags. Це окрема функція яка дозволяє виконати пошук найкращих тегів базуючись на описі завдання. Ці теги будуть виступати як класифікаційні для описаної задачі.

Отже базуючись на формулі 3.6 і рисунку 3.2 алгоритм процесу лише відбору і пошуку потрібних тегів скорочений та представлений на рисунку 3.3. Функція choose most used tags виконується на кінцевому етапі і повертає масив найкращих тегів.

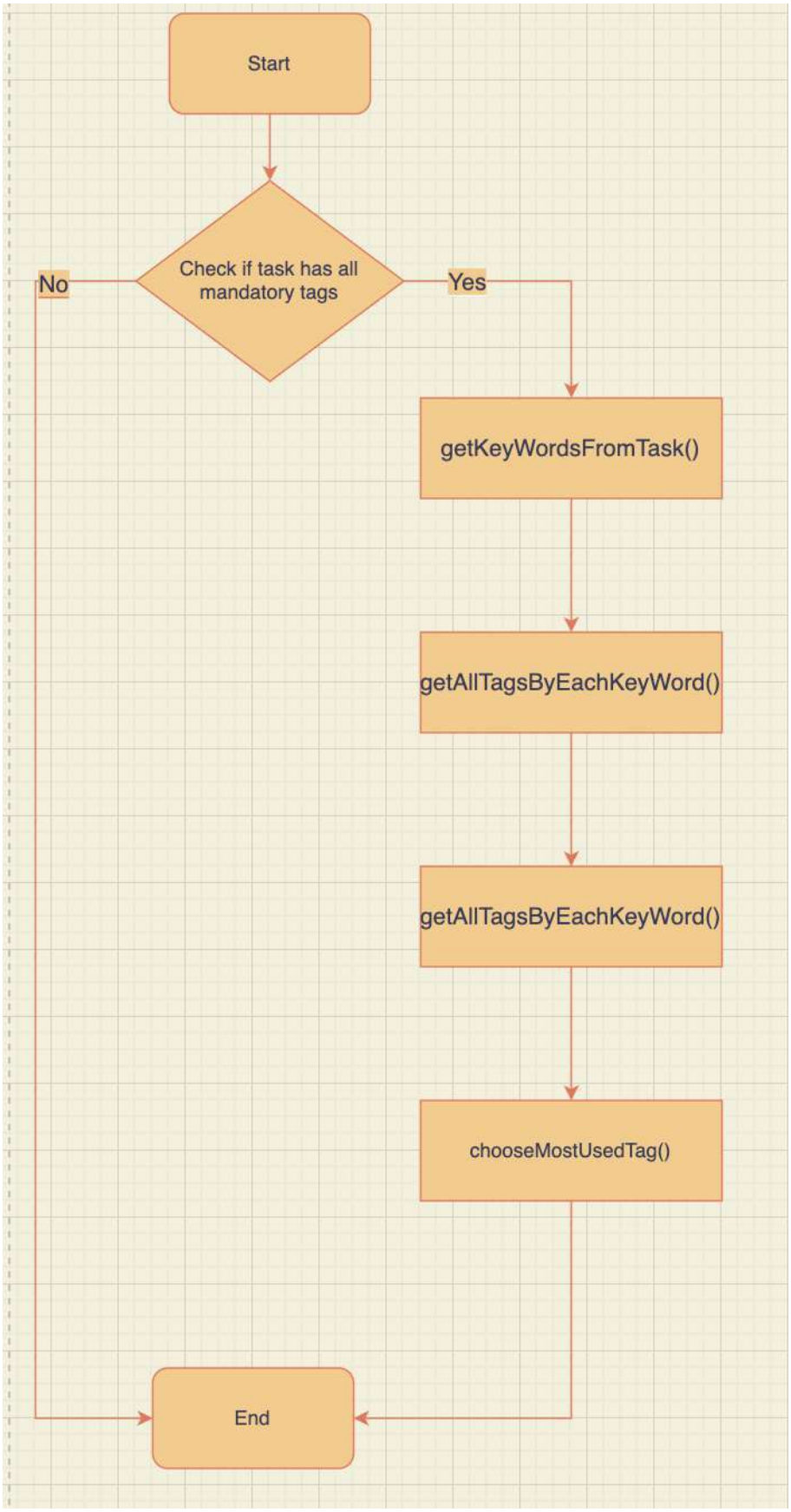


Рисунок 3.3 – Загальний алгоритм класифікації вибору найкращих тегів базуючись на описі завдання

3.2 Алгоритм розподілення задач

Для того щоб реалізувати алгоритм який буде займатися розподіленням задач між користувачами потрібно на першому етапі створити систему рейтингів. До системи рейтингів буде включено такі параметри:

1. Кількість задач виконано користувачем по всіх тегах які належать до групи тегів.
2. Кількість виконаних задач по всіх підгрупах тегів яка були визначені відповідно до опису задачі.
3. Загальна кількість задач які мають статус To do для кожного із учасників у рейтингу.
4. Загальна кількість задач які мають статус Next для кожного із учасників у рейтингу.
5. Загальна сума всіх оцінок часу на виконання кожного завдання які мають статус To do і Next.
6. Загальна сума оцінок які вистачити всі користувачі один одному для кожного окремого навичку.
7. Рейтинг по S(r). Це значення яке було розраховано за формулою 3.3 для кожного учасника рейтингу.
8. Рейтинг по M(r). Це значення яке було розраховано за формулою 3.7 для кожного учасника рейтингу.
9. Рейтинг по R(r). Це значення яке було розраховано за формулою 3.11 для кожного учасника рейтингу.
10. Рейтинг по D(r). Це значення яке було розраховано за формулою 3.15 для кожного учасника рейтингу.
11. Також сума по всіх решти тегах та навичках які є присутні у задачі після того як було виконано авто класифікацію задачі.

Тобто на першому етапі ми генеруємо один головний рейтинг по кількості виконаних задач по тег групах. На наступному кроці потрібно скласти під рейтинги із вище перерахованих елементів. Наступним крок введемо такі поняття як preferred

tags not і preferred tags. Кожен користувач на своїй персональні сторінці повинен буде мати можливість налаштовувати із якими тегами він би хотів працювати, а з якими ні. Це зроблено для того щоб користувач мав можливість розвиватися у різних напрямках. Також це такий підхід буде допомагати користувачу залишатися більш продуктивним у довшому періоді часу, і дозволить користувачу уникнути такого явище як ”вигорання”. Але для урахування бізнес процесу та логіки коефіцієнт впливу тегів яким би користувач надає перевагу і коефіцієнт впливу тегів з якими користувач не хотів би працювати на загальний рейтинг буде розраховуватись кожен раз під час процесу автоматичного розподілення задачі. Це буде зроблено для того щоб, не було випадку коли наприклад уся команда перейшла на проект і всі користувачі були на цьому проекті вибрали тег проекту і в меню налаштування обрали його як тег із якими не хотів би працювати, то щоб система базуючись на історії виконання задач командою на цьому проекті могла точніше визначити що задачу із тегом проекту потрібно віддати лише тій людині яка входила до команди яка працювала над проектом. А не обрала б користувача який має найкращий рейтинг саме по навичках або по групі навичок.

Для розрахунку впливу preferred tags використаємо формулу 3.7.

$$preferred = \prod_i 10 \cdot \{P_i\}/100, \tag{3.7}$$

де P(i) – це скільки разів зустрічається тег у завданні із тегами які є у користувача у полі preferred tags. Для розрахунку впливу preferred tags використаємо ту саму формулу 3.7.

Для того щоб розрахувати саме вплив коефіцієнта саме на оцінку із її зменшенням або збільшенням використаємо формулу 3.8.

$$mark = total\ mark + (preferred - not\ preferred), \tag{3.8}$$

де total mark – це оцінка користувача у результуючому рейтингу.

Наступним коефіцієнтом який також має свій вплив на загальну оцінку у загальному рейтингу це вплив часу на виконувані задачі. Тобто сума всіх оцінок уже існуючих задач із статусами To do і Next. Базуючись на цій цифрі ми визначаємо п’ять основних часових проміжків, це: 360, 720, 1440, 4320, 10 080.

Отже визначившись із базовими часовими проміжками нам необхідно зменшити позицію користувача у рейтингу на певний процент. Цей процент визначається в який саме часовий проміжок потрапляє сума загальних оцінок користувача. У випадку якщо сума оцінок усіх задач потрапляє у декілька часових проміжків оцінку в рейтингу по черзі для кожного проміжку потрібно зменшити. Уточненням також є те що потрібно зменшувати кожен раз не першу оцінку яка була до зменшення у рейтингу а вже зменшену на якийсь заданий процент, це у випадку коли загальна оцінка задач потрапляє у декілька визначних часових проміжків. Процент на який потрібно зменшити оцінку вибирається у порядку зростання довільним чином. На першому етапі було обрано такі значення: 2, 5, 10, 15, 20 відсотків.

У випадку якщо сума загальної оцінки наявних задач користувача менша за шість годин то зменшення оцінки не відбувається.

Для фінального кроку потрібно отриманий загальний рейтинг відсортувати по оцінках і обрати найкращого кандидата на виконання поставленої задачі. У випадку якщо ж у кандидатів однакові оцінки то вибирається той кандидат у якого час загальної оцінки усіх задач зі статусом To do і Next найменший. Якщо ж і час виконання найменший то обирається перший зі списку. Якщо ж у системі рейтингів відсутній жоден кандидат то система повертає значення false і виводить повідомлення що кандидатів на виконання поточної задачі не існує. Але такий випадок малоймовірний оскільки усі користувачі які знаходяться в межах одного працюючого місця в додатку беруть участь у створені рейтингів базуючись на їхньому досвіді перебування на проекті, який вже є створений у дата сеті або ж проект ігнорується якщо цей проект наприклад новий і команда тільки починає працювати над новим проектом. В такому випадку користувачам потрібно в ручному режимі вказати системі щось конкретно ця задача належить до нового

проекту, під час створення та опису задачі. Тобто у автономній та інтегрованій системі має бути також реалізований механізм виправлення або ж перенавчання системи для конкретного кейсу. Такий механізм дозволить підвищити точність прогнозування методу, а також в такий спосіб можна збільшити надійність розроблювальної системи та її автономність.

Отже, розроблений підхід визначення найкращого кандидата і використовуючи при цьому систему рейтингів, а також вплив preferred tags і not preferred tags і вплив загального часу виконання уже наявних задач із статусом To do і Next вказує нам на те, що метод лінійного програмування є більш зручним і оптимальнішим методом для розв’язання задачі, ніж метод генетичного алгоритму. При цьому для цільової функції можна використати формулу, яка і враховує три базових параметри для виконання пошуку найкращого кандидата для виконання задачі.

Алгоритм який відповідає за зменшення оцінка учасника у загальному рейтингу базуючи на сумі усіх оцінок задач зі статусом To do і Next можна побачити на рисунку 3.4

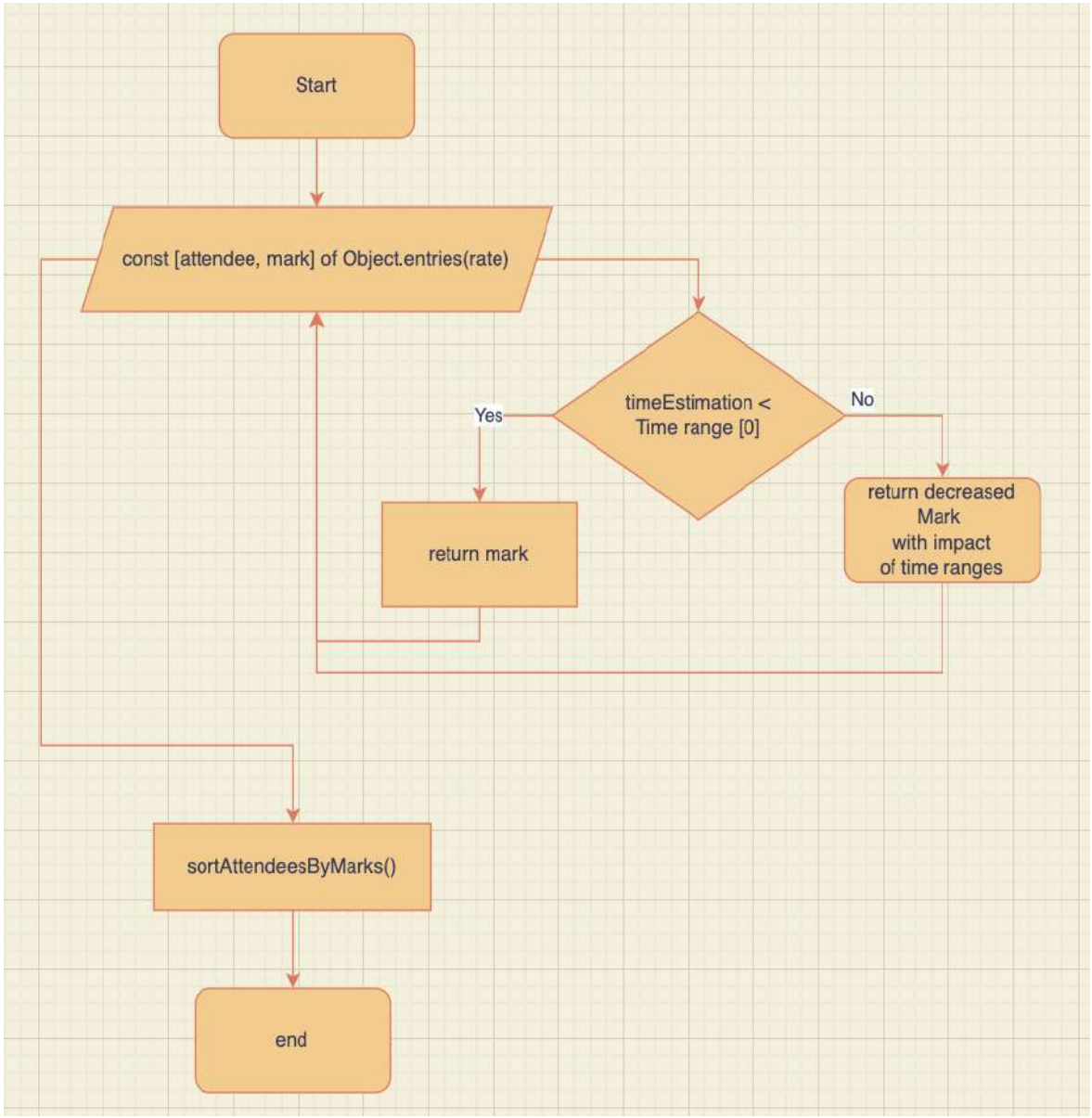


Рисунок 3.4 – Блок схема алгоритму для зменшення оцінки в загальному рейтингу

Функція decrease mark with impact of time ranges має бути реалізована і її блок схема представлена на рисунку 3.5.

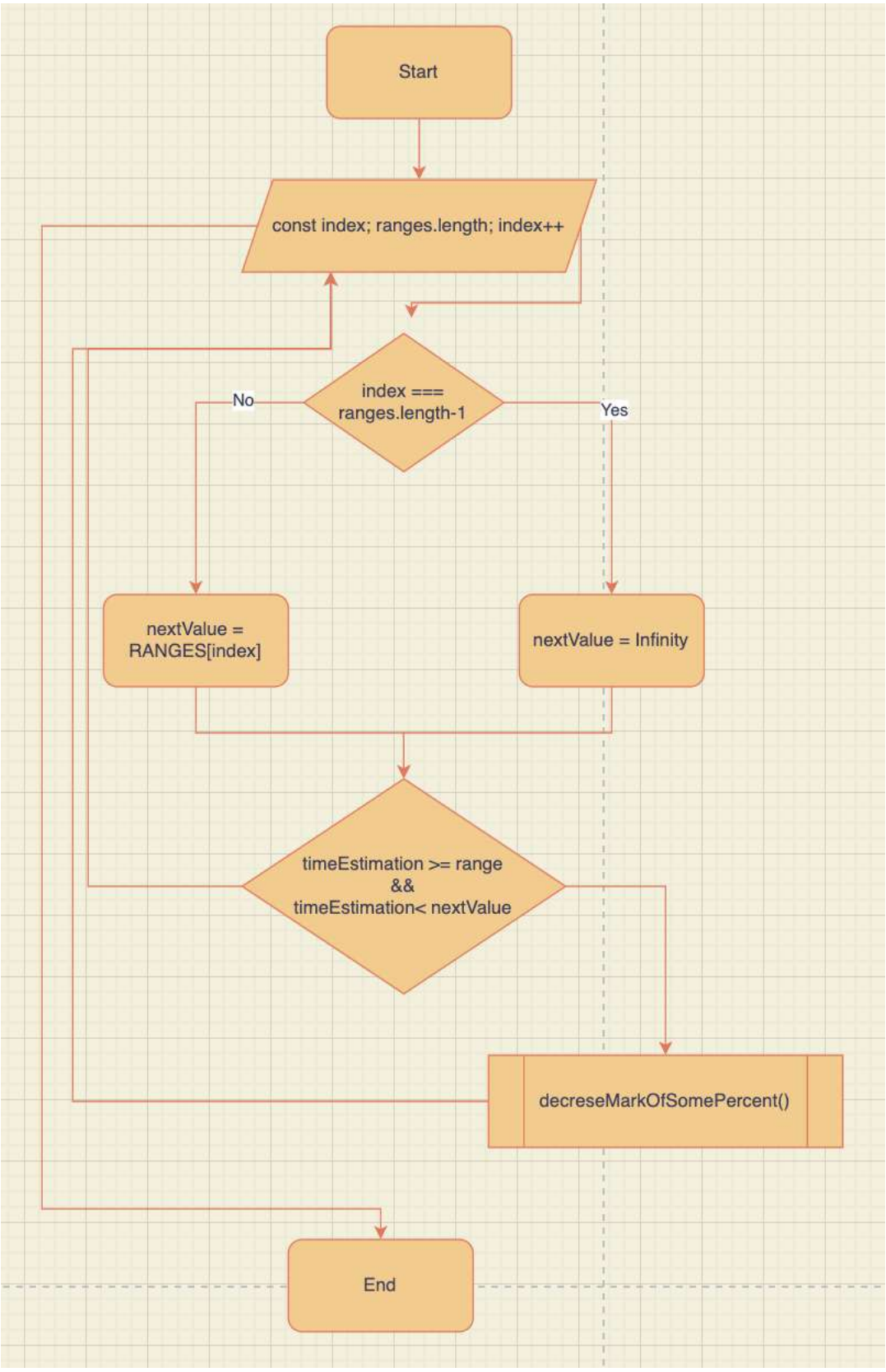


Рисунок 3.5 – Блок схема зменшення оцінки в рейтингу враховуючи усі проміжки часу в які входить сума усіх оцінок задач

Для реалізації алгоритму із впливом preferred і not preferred tags була розроблена блок схема на рисунку 3.6.

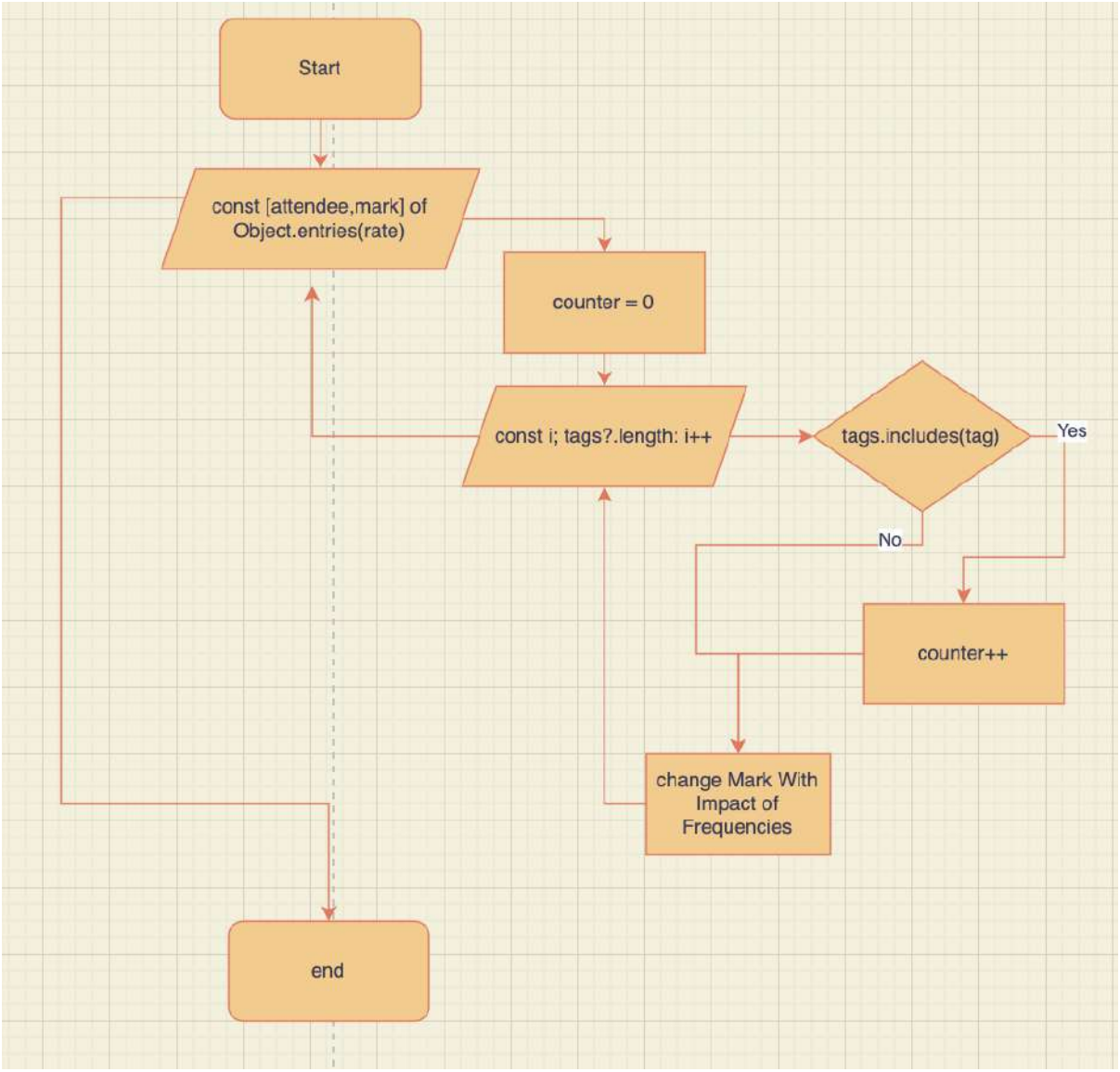


Рисунок 3.6 – Блок схема зменшення оцінки в рейтингу враховуючи усі preferred і not preferred tags

У функції change mark with impact of frequencies реалізований механізм який зменшу або збільшує в циклі оцінку користувача в рейтингу стільки разів, скільки зустрічалися preferred або not preferred tags. При цьому зменшуючи не першу оцінку на процент, а кожну наступну або зменшену або збільшену оцінку. Вага коефіцієнта для збільшення чи зменшення становить десять відсотків і було визначена експериментальним шляхом, використовуючи спосіб який довільно

обирав число на проміжку та визначав наскільки обраний коефіцієнт має вплив на загальний результат. Зі всіх результатів коефіцієнт в десять відсотків був найоптимальнішим рішенням для поставленої задачі.

3.3 Проектування програмного забезпечення для вирішення розподілення та оцінювання задач в процесі розробки програмного забезпечення

Для того щоб реалізувати алгоритм який буде поєднувати в собі механізм автоматичної класифікації задач, а також механізм автоматичного визначення найкращого кандидата, потрібно також розробити механізм який буде займатися підготовкою та генерацією отриманих даних із опису задачі в потрібний нам дата сет який буде використовуватися двома наступними алгоритмами. Також необхідно виконати проектування механізму комунікації та транспортування та зберігання даних між станами функцій та алгоритмів.

Для початку потрібно розібратися в який саме спосіб буде відбуватися генерація дата сету. На етапі коли користувач створює задачу а також опис цієї задачі ми маємо текст опису задачі із всіма необхідними деталями. Для того щоб можна було отримати потрібні нам відношення і виконати автоматичну класифікацію задач нам потрібно розбити текст задачу на окремі не залежні частини. Тобто потрібен механізм який дозволить видалити усі зайві частини опису і отримати лише конкретні зв’язки між словами і першу початкову думку задачі. Зазвичай для рішення такого роду задач використовують підхід із використанням NLP.

NLP – або ж обробка природньої мови це підхід який дозволяє, базуючись на теоретичні частині та певному наборі технологій вирішити проблему взаємодії людських вербальних та невербальних комунікативних актів та комп’ютерних систем. Зазвичай для розуміння людської природньої мови потрібно багато знань для системи про навколишнє середовище та які існують можливості взаємодіяти із середовищем. Головні завдання які вирішує NLP це:

1. Видобування даних. Наприклад пошук закономірностей.

2. Добування даних. Тобто отримання семантичної інформації тексту.
3. Синтез мовлення.
4. Машинний переклад.
5. Спрощення тексту.

Також існують різні підходи для вирішення завдань. Наприклад такі підходи: Статистичний, Лінгвістичний, Символічний, умовні випадкові поля, метод допоміжних векторів тощо.

Також для метод обробки природньої мови використовується на різних рівнях структури мови. Існує шість таких рівнів. Це:

1. Прагматичний.
2. Семантичний.
3. Синтаксичний.
4. Лексичний.
5. Морфологічний.
6. Фонологічний.

Для вирішення нашої задачі нам потрібно виконувати обробку мови на рівні граматичний аналізу. Оскільки на цьому рівні відбувається аналіз слів для розуміння граматичної структури речення. Але використання NLP для вирішення такої задачі це може бути занадто. Оскільки нам необхідно мінімізувати час виконання всього алгоритму. Також на стадії розробки MVP проекту, це затратно по ресурсам, а також не необхідно мати певний рівень експертизи для виконання цієї задачі. Тому я пропоную використати підхід який називається keyword extractor. Для розробки MVP проекту цього більш ніж достатньо. Оскільки задачі описуються виключно англійською мовою це спрощує нам задачу. Алгоритм keyword extractor дозволяє отримувати ключові слова із речення використовуючи при цьому стоп слова.

Отже використовуючи метод keyword extractor ми отримаємо усі ключові слова із опису задачі. Наступним кроком нам необхідно взяти всі отримані ключові слова, а також уже існуючі теги та користувачів які мають виконувати поточну задачу, якщо такі є, і сформувати граф відношень, рисунок 3.7, кожного з кожним

для того щоб в майбутньому ми могли використати наявну інформацію для механізму автоматичної класифікації задачі та для механізму автоматичного розподілення задач. Також після виконання алгоритму нам потрібно повторити таке генерування для того щоб система могла запам’ятати і добавити до своїх ваг ще одну одиницю із правильним передбаченням. У випадку якщо завдання було видалення для всіх вершин потрібно зменшити на одиницю їхнє значення. У випадку якщо завдання було змінене для стану до потрібно зменшити на один для стану після додати один.

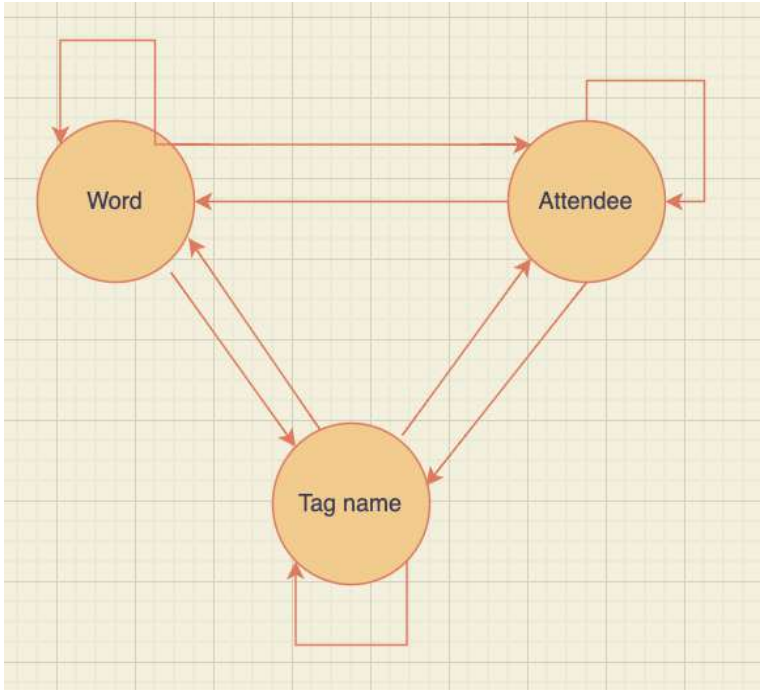


Рисунок 3.7 - Граф відношень кожен із кожним. До вершин графу входять слово, тег, користувач

А для того щоб виконати планування виконання усіх алгоритмів за одну ітерацію необхідно виконати такі послідовні кроки:

1. Згенерувати дата сет із базового опису задачі. Із існуючим класифікаційними тегами та користувачами які будуть виконувати задачу якщо такі є.
2. Виконати метод для класифікації задачі.
3. Базуючись на отриманих класифікаційних тегах виконати вибір найкращого кандидата на виконання задачі.

4. Закріпити із новими вагами дата сет.

Функціонування такої системи зображено на рисунку 3.8.

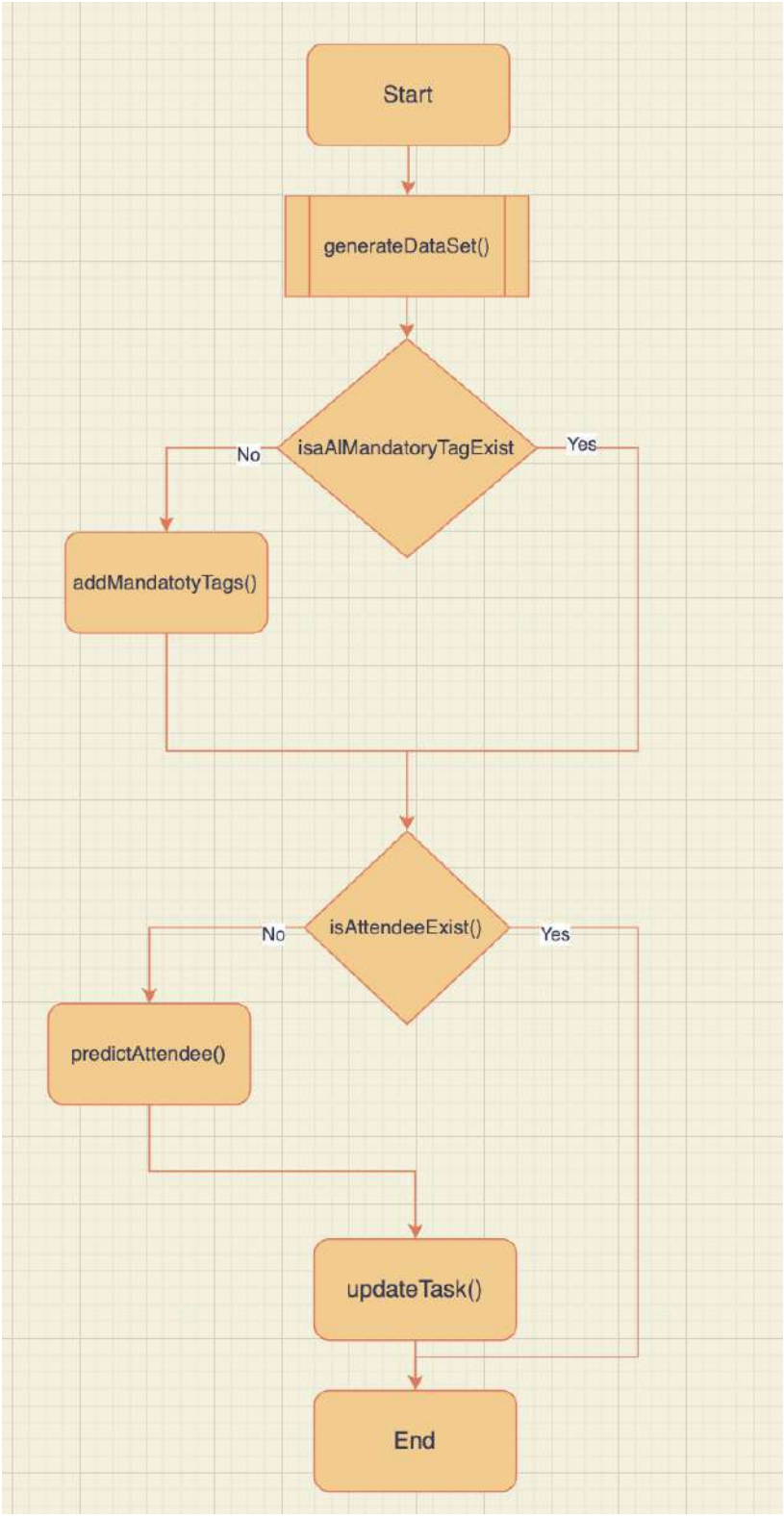


Рисунок 3.8 - Блок схема алгоритму виконання усіх необхідних процесів у одній ітерації

3.4 Висновки

В результаті проведеного аналізу та планування і розробки алгоритмів та методів розподілення та класифікації задач під час розробки програмного забезпечення було виявлено, що існуючі підходи до системи розподілення та класифікації задач мають такі як свої переваги та недоліки.

Було розроблено окремо метод для класифікації задач базуючись на повному описі поставленої задачі. В цьому методі класифікаторами виступають теги. Існує п’ять базових групи тегів які обов’язково мають бути присутніми у створеній задачі. У випадку якщо немає можливість добавити потрібні теги добавляються зарання визначені теги, окрім тег групи проекту. Головною особливістю є те що довелося відмовитись від базового представлення дерев прийняття рішень і використати систему графів які мають відношення багато до багатьох для того щоб отримати історію відношень і частоти використання пар слів і тегів навичок та проектів. Що в свою чергу допоможе уникнути помилок ще на етапі аналізу тексту поставленої задачі в процесі розробки програмного забезпечення. Даний метод зможе з більшою точністю виконувати класифікацію задач, навіть у тих випадках коли історія використання пар слів і тегів відсутня, або ж існує, але відношення занадто слабкі у базовому графі який використовується ще на першому етапі розпізнавання цих відношень.

Також було розроблено окремий метод який відповідає за розподілення задач. Враховуючи те що генетичний алгоритм потребує зарання визначеної кількості етапів еволюції або ж потрібно враховувати обмеження по часу виконання алгоритму, було прийнято рішення використати у методі для розподілення задач лінійне програмування, де по факту цільовою функцією була створена система рейтингів. Оскільки зазвичай машини мають обмежений запас ресурсів і також повинні мати високу пропускну здатність, ми не можемо дозволити функцію визначення найкращого кандидата просто зависнути. Це ще один чинник який спонукав обрати метод лінійного програмування, використовуючи при цьому лише один етап для пошуку і це є головною

особливістю розробленого методу. Для того щоб підвищити точність, а також врахувати інтереси бізнесу, в загальну формулу пошуку найоптимальнішого кандидата також було додано вплив сума загальних оцінок усіх задач які мають статус To do і Next. А також вплив тегів або ж групи тегів. Для збільшення оцінки використовується кількісне співпадіння тегів із якими користувач хоче працювати, а для зменшення оцінки в рейтингу використовується кількісне співпадіння тегів із якими користувач не хоче працювати.

Було визначено основні етапи проектування інтегрованої автоматичної системи. Розроблені механізми дозволяють підвищити стабільність. Покращити комунікацію, збільшити швидкість розробки, а також спросити та автоматизувати процес менеджменту задач в середині команди під час розробки програмного забезпечення.

Отже для реалізацію алгоритмів потрібно обов’язково реалізувати такі методи та додаткові основні функції:

1. Метод який буде діставати із опису задачі важливі дієслова та іменники.
2. Метод який буде генерувати дата сет. А також метод який буде виправляти дата сет у випадку коли передбачення системи було не зовсім точним.
3. Метод який буде шукати найсильніші відношення у дата сеті між ключовими словами у описі задачі і тегами. Також враховуючи при цьому тег групу.
4. Метод який буде шукати найкращого кандидата для виконання поставленої задачі базуючись на тегах які були відібрані етапом раніше.
5. Метод який поєднає в собі усі попередні кроки і буде виконувати на створення або редагування поставленої задачі.

4 РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ВИРІШЕННЯ ЗАДАЧІ

4.1 Програмна реалізація системи автоматичного розподілення та класифікації задач

Під час виконання кваліфікаційної роботи було реалізовано MVP прототип веб – додатку, який являє собою систему розподілення та класифікації задач під час розробки програмного забезпечення. Програмна реалізація MVP прототипу включає в себе реалізацію базової клієнт – серверної архітектури. Було створено два окремих додатки. Тобто логіка взаємодії із користувачем і логіка виконання алгоритмів була розділена на дві частини. Нас же більше цікавить частина де було реалізовано алгоритми автоматичної класифікації задач та алгоритм вибору найкращого кандидата на виконання завдання.

Опис основних функцій програмного забезпечення подано в зведеній таблиці 4.1.

Таблиця 4.1 – Опис основних функцій ПЗ

№ п.п	Назва функції	Опис функції
1	handleAutomaticallyAddTagToTask()	Функція яка є точкою входу для головного алгоритму автоматичного визначення тегів. Де параметром функції є опис до задачі.
2	checkIfTaskHasAllMandatoryTags(tags)	Ця функція допомагає перевірити, чи завдання має всі обов’язкові теги
3	getKeywordsFromTask(title)	Ця функція отримує всі ключові слова з опису задачі.
4	getAllTagsByKeywords(keywords)	Ця функція отримує всі теги, які використовувалися з кожним ключовим словом

Продовження таблиці 4.1 – Опис основних функцій ПЗ

№ п.п	Назва функції	Опис функції
5	getMostUsedTagsForEachMandatoryTags()	Ця функція допомагає автоматично додавати тег до завдання
6	chooseMostUsedTags()	Ця функція допомагає вибрати найкращі теги для базового завдання на обов’язкових тегах
7	handleCRUDDataSetForAutoAssign()	Ця функція дозволяє згенерувати потрібний нам дата сет на всі потрібні івенти
8	onWriteTask(task)	Функція яка є точкою входу для функції всередині якої відбуваються автоматичне визначення найкращого кандидата, автоматична класифікації задачі, а також генерація дата сету.
9	onAutomaticallyAssign()	Функція яка є точкою входу і виконує автоматичне визначення найкращого кандидата.
10	handleAutomaticallyAssign()	Функція яка виконує автоматичне визначення найкращого кандидата.
11	getTypeOfSkillsByTagGroups()	Ця функція отримує всі теги, пов’язані з групою тегів, і повертає два значення, де перше з тегів групи імен, а друге — це об’єкт, де ключі — це назва групи тегів, а значення — це теги, які містять тег групи
12	getAttendees()	Ця функція витягує всіх учасників f

Продовження таблиці 4.1 – Опис основних функцій ПЗ

13	getAttendeesTagsFromRDB()	Ця функція отримує всі теги, пов’язані з учасником із real time database.
14	filterPollForCreatingRate()	Wя функція допомагає фільтрувати опитування та використовувати лише повні дані для розрахунку рейтингу.
15	transformAttendeeDataForCreatingRate()	Ця функція допомагає трансформувати дані учасників і дані опитування та використовувати лише повні дані для розрахунку рейтингу.
16	createRate()	Ця функція допомагає створити ставку для кожного параметра в attendee and related data
17	transformDataBeforeCalculateTotal()	його функція допомагає трансформувати дані та змінювати місцями ключі та значення в глибокому об’єкті, після перетворення ключі будуть позначками, а значення будуть масивами учасників, які мають однакові позначки
18	updateTotalMarkAttendeesBasicOnPreferredTags()	Ця функція допомагає оновити базові оцінки відвідувачів на теги, яким вони віддають перевагу перевагу
19	updateTotalMarkAttendeesBasicOnNotPreferredTags()	Ця функція допомагає оновити базові оцінки відвідувачів на теги, яким вони не віддають перевагу
20	transformRateForChoosingAttendee()	Ця функція допомагає трансформувати загальну ставку перед вибором основного учасника

Кінець таблиці 4.1 – Опис основних функцій ПЗ

№ п.п	Назва функції	Опис функції
21	decreaseMarkOfAttendeeInRateBasicOnTheirEstimationForToDoTasks()	Ця функція допомагає знизити оцінку відвідувачів залежно від суми часу оцінки для всіх завдань зі статусом To_do
22	checkIfKeysInObjectHaveSameMarks()	Функція викликається для передачі керування функції, яка відповідає за вибраний елемент зі списку
23	chooseAttendee()	Ця функція допомагає вибрати учасника залежно від його оцінок

Уся система повинна працювати наступним чином. Користувач із будь якою роллю повинен через створений нами інтерфейс, тобто веб застосунок, повинен мати можливість зайти у застосунок і у формі створення завдання створити детальний опис задачі і зберегти опис задачі в базі firebase firestore. Наступним кроком firebase functions слухає зміни в базі даних і відправляє на окремий API endpoint усі дані задачі а також тип події. Типи подій можуть бути: створення, зміна, видалення. В цьому API endpoint система отримує дані і запускає функцію onWriteTask. В цій функції на першу чергу запускається функція яка відповідає за створення дата сету на базі вже існуючого опису. Наступним кроком запускається функція яка автоматично визначає потрібні теги і повертає їх. Далі вже раніше визначені теги а також опис задачі потрапляють у функцію яка шукає найкращого кандидата на виконання поставленої задачі. Ця функція використовує дата сет а також дані із оцінювання користувачів і дані із preferred and not preferred tags та повертає найкращого кандидата і на останньому етапі виконується подія оновлення задачі із визначеними усіма обов’язковими тегами та найкращим кандидатом. Після цього також необхідно знову запустити функцію яка оновить дата сет із новими даними.

Система оцінювання користувача по всі раніше визначених навичках зображена на рисунку 4.1. На цій сторінці із формою в користувача існує можливість оцінити іншого користувача оцінка від одного до десяти, а також реалізована система не визначеної оцінки. Якщо користувач не знає як оцінити навичку користувача із ряду існуючих причин то коли він вибере цю опцію, то значення в розрахунках для цієї навички просто не будуть враховуватись при калькуляції додаткових значень. Також реалізований механізм коли взагалі не можливо оцінити користувача по всім навичкам.

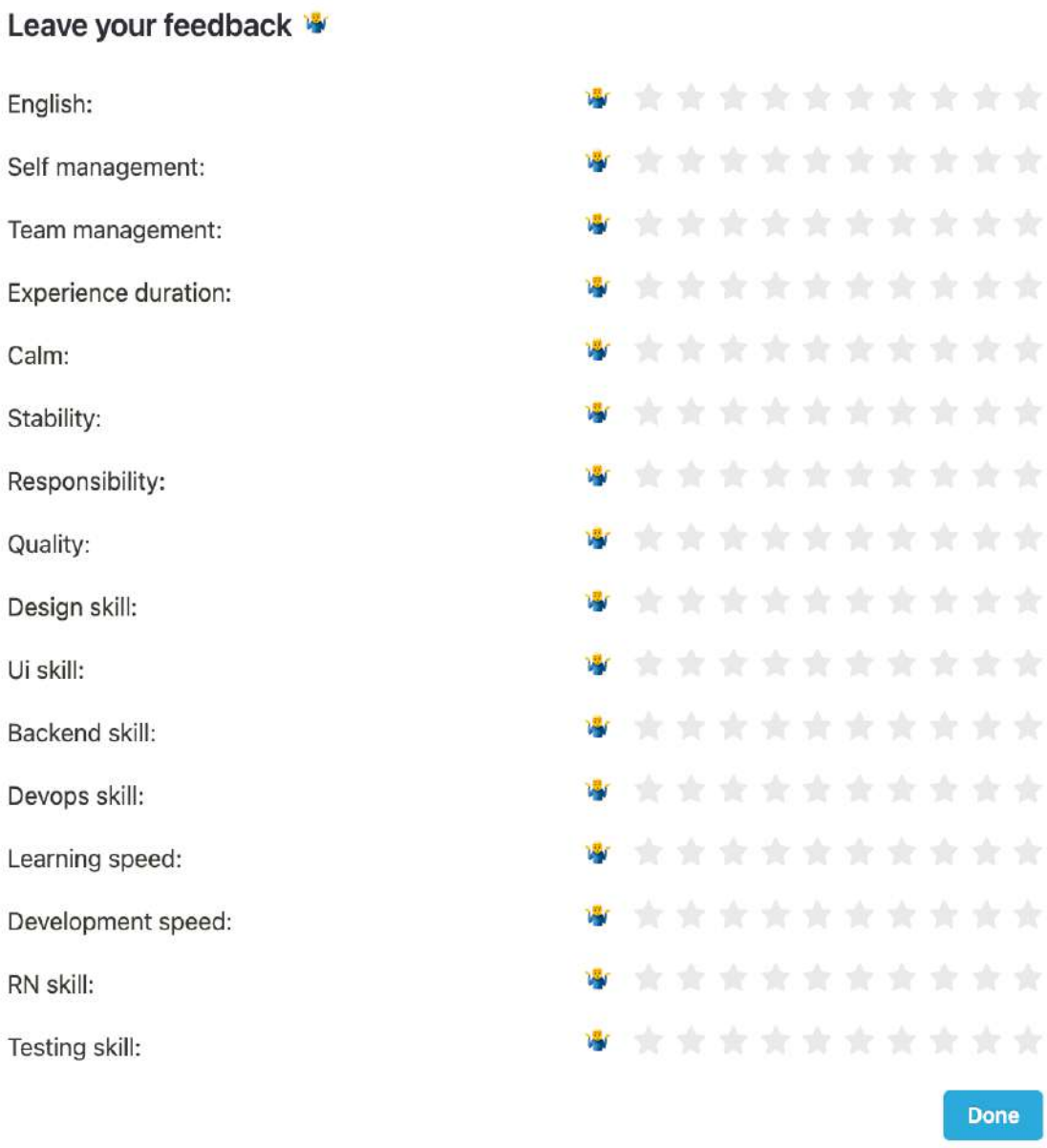


Рисунок 4.1 – Реалізація механізму оцінювання користувачів іншими користувачами

Система в меню для вибору тегів із якими людина хотіла б працювати а також теги із якими не хотіла б зображена на рисунку 4.2

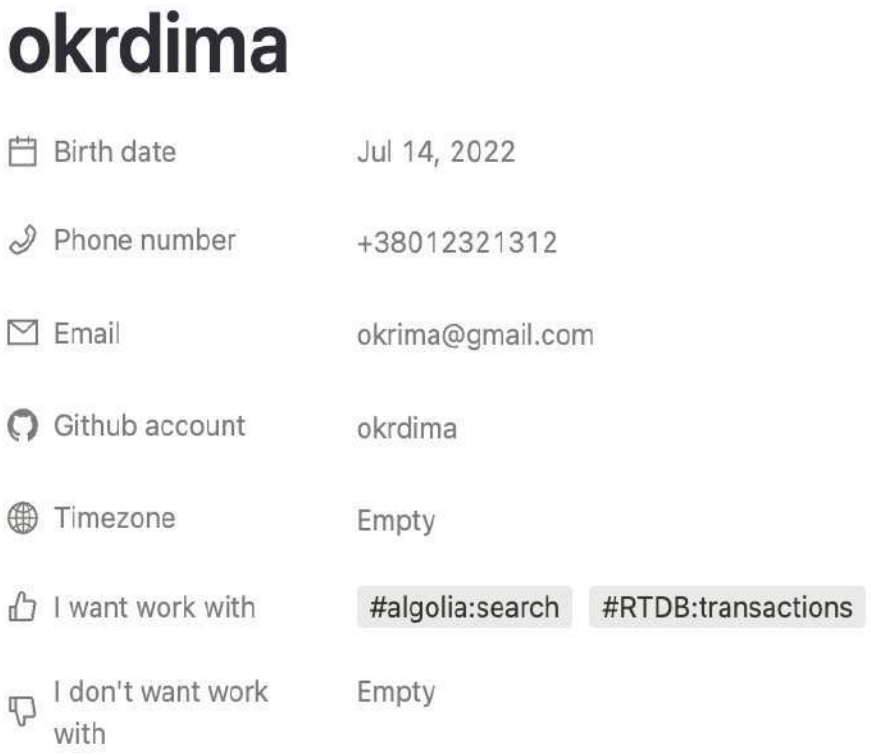


Рисунок 4.2 – Реалізація меню вибору preferred і not preferred tags на сторінці користувача

Схема того як уся система працює і зв’язана між собою зображена на рисунку 4.3

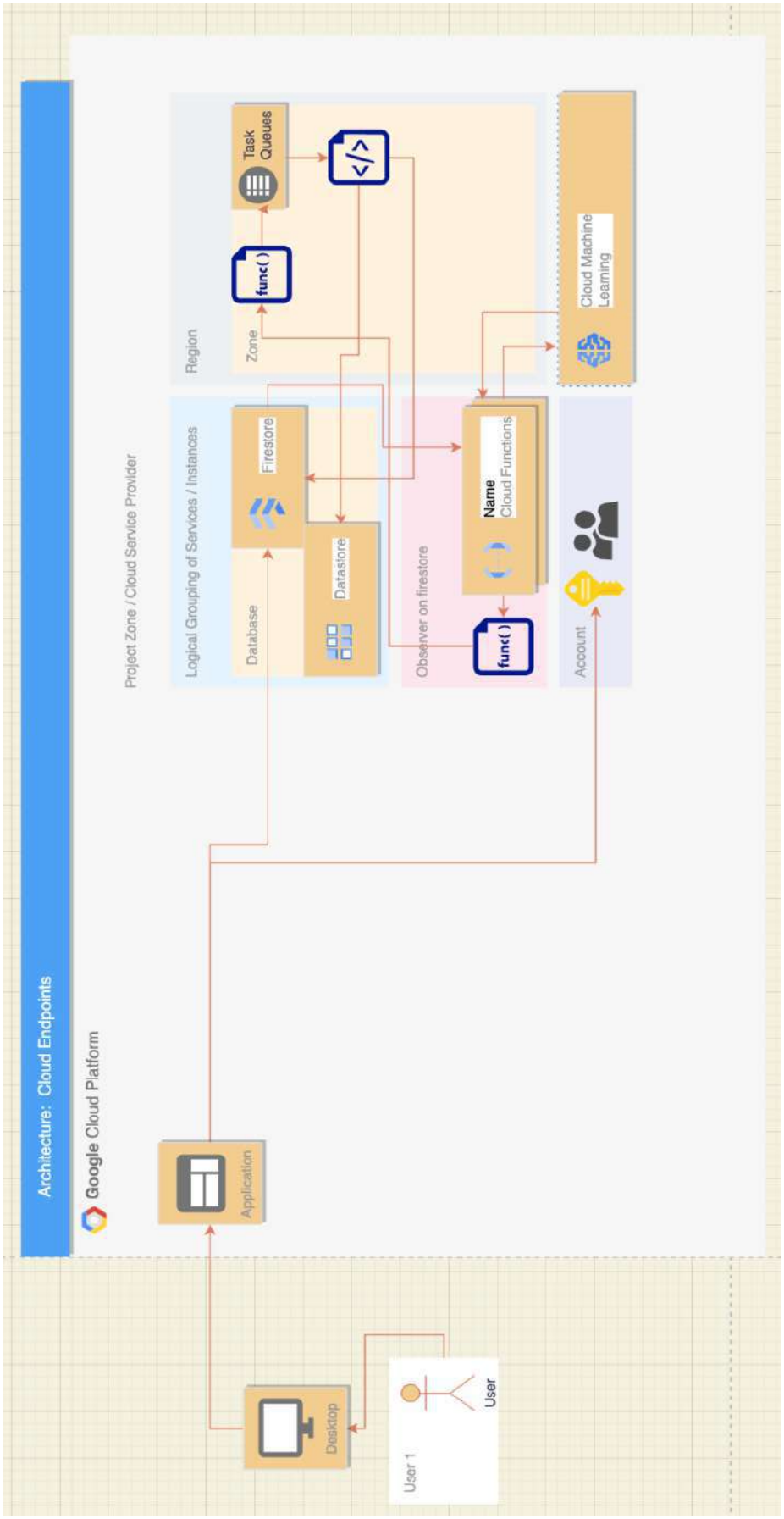


Рисунок 4.3 – Реалізація взаємодії системи на платформі GCP

4.2 Обґрунтування стеку технологій та сервісів для реалізації поставленого завдання

Для реалізації поставлених задач було вирішено обрати MERN стек технологій. Було обрано цей стек оскільки в стекові використовується мова програмування java script. Стек MERN — це набір технологій, які дозволяють швидше розробляти програми. Його використовують розробники по всьому світу. Основною метою використання стеку MERN є розробка програм лише за допомогою JavaScript. Це пояснюється тим, що всі чотири технології, які складають стек технологій, базуються на JS. Таким чином, якщо хтось знає JavaScript (і JSON), можна легко керувати бекендом, інтерфейсом і базою даних. MERN Stack — це компіляція чотирьох різних технологій, які спільно розробляють динамічні веб-програми та веб-сайти. До складу MERN входять:

1. MongoDB.
2. ExpressJS.
3. NodeJS.
4. React JS.

Але було вирішено змінити базу даних у якій я буду працювати із MongoDB на Firebase.Існує чотири компоненти стеку MERN. Давайте обговоримо кожен з них по черзі.

1. Першим компонентом є MongoDB, яка є системою керування базами даних NoSQL. Але як вже було зазначено що базу даних було змінено на Firebase яка також є NoSQL.
2. Другим компонентом стеку MERN є ExpressJS. Це базова основа веб-додатків для NodeJS.
3. Третім компонентом є ReactJS, бібліотека JavaScript для розробки UI на основі компонентів UI.
4. Останнім компонентом стеку MERN є NodeJS. Це середовище виконання JS, тобто воно дозволяє запускати код JavaScript поза браузером.

Firebase — це платформа для розробки додатків, яка допомагає створювати та розвивати додатки та ігри, які подобаються користувачам. Підтримується Google і її використовують мільйони компаній у всьому світі. Firebase це СУБД NoSQL, де дані зберігаються у формі документів, що мають пари ключ-значення, подібні до об’єктів JSON. Firebase дозволяє користувачам створювати бази даних, схеми та таблиці. Він пропонує оболонку FirebaseSDK, яка надає інтерфейс JS для видалення, запиту та оновлення записів. Інтерфейс Firebase зображено на рисунку 4.4.

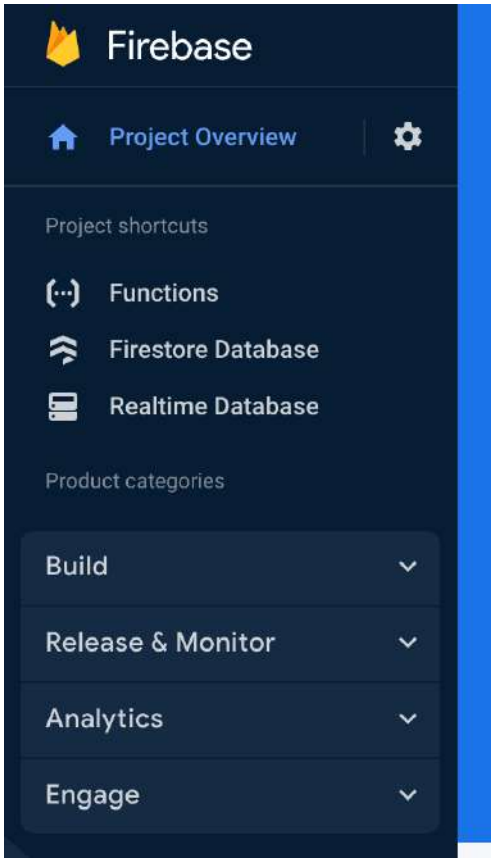


Рисунок 4.4 – Навігаційне меню СУБД Firebase

ExpressJS — це фреймворк NodeJS, який спрощує написання базового коду. Це позбавляє вас від створення кількох модулів Node. Для підтримки точності коду ExpressJS пропонує ряд проміжного ПЗ.

ReactJS — це бібліотека JS, яка дозволяє розробляти інтерфейс користувача для мобільних додатків і SPA. Це дозволяє кодувати JavaScript і розробляти

компоненти інтерфейсу користувача. Бібліотека JS використовує віртуальний DOM, щоб робити все.

NodeJS — це середовище виконання JavaScript із відкритим кодом, яке дозволяє користувачам запускати код на сервері. Він поставляється з менеджером пакетів вузлів або npm, що дозволяє користувачам вибирати з широкого вибору модулів або пакетів вузлів. Розробка на основі Chrome JavaScript Engine дозволяє Node швидше виконувати код.

Є багато вагомих причин використовувати стек MERN. Наприклад, це дозволяє створити 3-рівневу архітектуру, яка включає інтерфейс, серверну частину та базу даних за допомогою JavaScript і JSON.

Firebase призначена для зберігання даних JSON у рідному вигляді. Усе в ньому, включаючи CLI та мову запитів, створено за допомогою JSON та JS. Система керування базами даних NoSQL добре працює з NodeJS і, таким чином, дозволяє маніпулювати, представляти та зберігати дані JSON на кожному рівні програми.

Express — це платформа на стороні сервера, яка обгортає HTTP-запити та відповіді та полегшує зіставлення URL-адрес із функціями на стороні сервера. Це ідеально доповнює фреймворк.

ReactJS, інтерфейсний фреймворк JS для розробки інтерактивних інтерфейсів користувача в HTML під час спілкування з сервером.

Оскільки дві технології працюють із JSON, дані передаються без проблем, що дає змогу швидко розробляти та легко налагоджувати. Щоб зрозуміти всю систему, вам потрібно розуміти лише одну мову, тобто JavaScript і структуру документа JSON.

Сучасні веб-розробники з достатнім досвідом роботи з React і JS віддають перевагу розробці інтуїтивно зрозумілих додатків за допомогою стеку MERN.

Як і в інших популярних веб-стеках, у MERN можна розробляти все, що завгодно. Тим не менш, він ідеально підходить для хмарних проектів, де вам потрібні інтенсивні JSON і динамічні веб-інтерфейси.

4.3 Результати експериментів, аналіз та оцінка ефективності методів автоматичної класифікації задач та системи автоматичного вибору найкращого кандидата

Для експериментального дослідження використовувався згенерований нами набір даних. Для створення набору даних за основу була взята структура графа, де кожен вузол має зв’язок з усіма іншими вузлами. Згенерована структура даних має вигляд як на рисунку 4.5. Варто звернути увагу, що так як ми маємо обмеження платформи де ключ в об’єкті не може бути стрічка яка містить символи “.”, “[”,”]”, було прийнято рішення використовувати ім’я користувача як ключ попередньо обробивши його за допомогою бібліотеки md5. Тобто використати хеш ім’я користувача. Для простої візуалізації прогнозу оптимального кандидата візьмемо для прикладу опис завдання - «Я, як супер адміністратор, повинен мати можливість створити форму для запрошення користувачів», і ніяких тегів кваліфікації додаватися не буде.

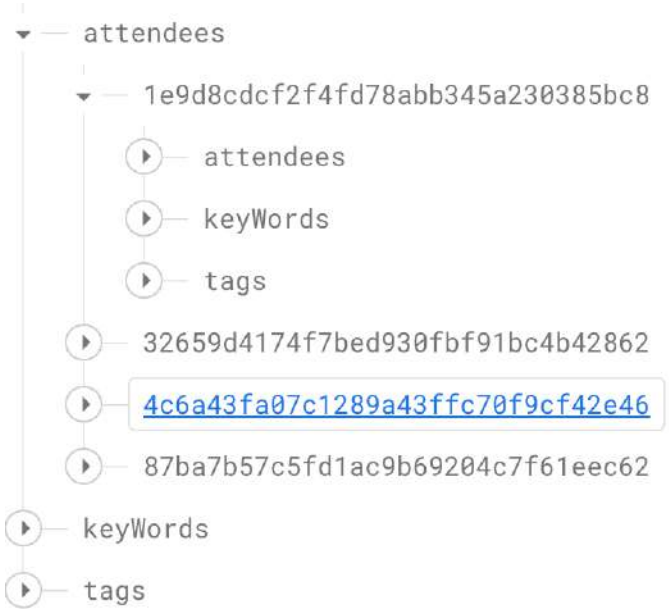


Рисунок 4.5 – Згенерована структура даних в real time database

На рисунку 4.5 ви можете побачити чітко структуровану сутність, де кожен ключ має ідентифікатор учасника, який хешується. У свою чергу, кожен учасник

має посилання на кожного учасника, включаючи себе. Структура також включає ключові слова, які також хешуються, описуваного завдання та всіх існуючих і використовуваних тегів у системі. Це допоможе в майбутньому, коли ми створимо найбільш сумісні та відповідні теги для опису завдання.

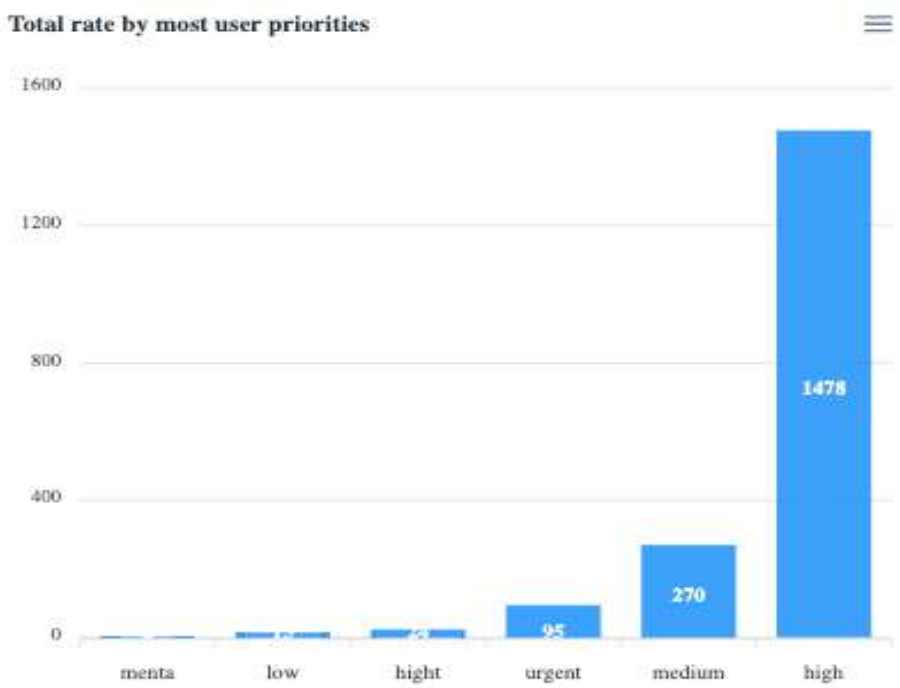


Рисунок 4.6 – Рейтинг за найбільш використовуваними пріоритетами

На рисунку 4.6 видно, що з усіх тегів, присутніх у структурі графа (height, low, urgent, medium), тег «high» є найбільш часто використовуваним для всіх слів, описаних у завданні «Я як супер адміністратор повинен мати можливість створити форму для запрошення користувачів».

Цей тег використано 1478 разів. Він має величезний відрив від тегу «medium», який має лише 270.

Це говорить про те, що тег «high» є, ймовірно, найоптимальнішим варіантом для описаного завдання.

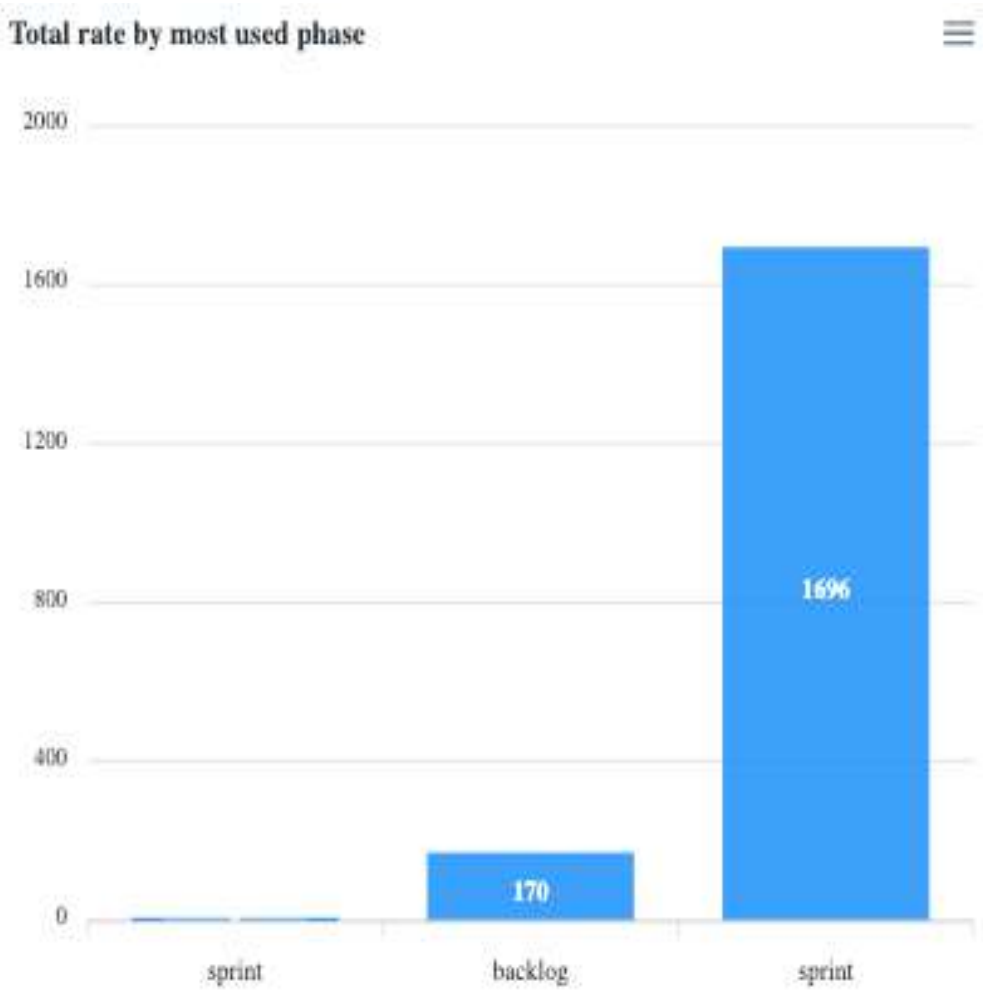


Рисунок 4.7 – Рейтинг за найбільш використовуваними фазами

На рисунку 4.7 видно, що з усіх тегів, присутніх у структурі графа (sprint, backlog), тег sprint найчастіше використовується для всіх слів, описаних у завданні «Я, як супер адміністратор, повинен вміти створювати форму для запрошення користувачів». Цей тег використано 1696 разів. Він має величезний відрив від тегу 'backlog', який має лише 170. Це говорить про те, що тег 'sprint' є, мабуть, найбільш оптимальним варіантом для описаного завдання. Далі з усіх тегів, присутніх у структурі графа, тег react найчастіше використовується для всіх слів, описаних у завданні «Я, як супер адміністратор, повинен вміти створювати форму для запрошення користувачів». Цей тег використано 1384 разів. Він має величезний відрив від тегу 'J/S', який має лише 430. Це говорить про те, що тег 'react' є, мабуть, найбільш оптимальним варіантом для описаного завдання. Далі з усіх тегів, присутніх у структурі графа, тег task найчастіше використовується для всіх слів.

Цей тег використано 1685 разів. Він має величезний відрив від тегу 'bug', який має лише 105. Це говорить про те, що тег 'task' є, мабуть, найбільш оптимальним варіантом для описаного завдання.

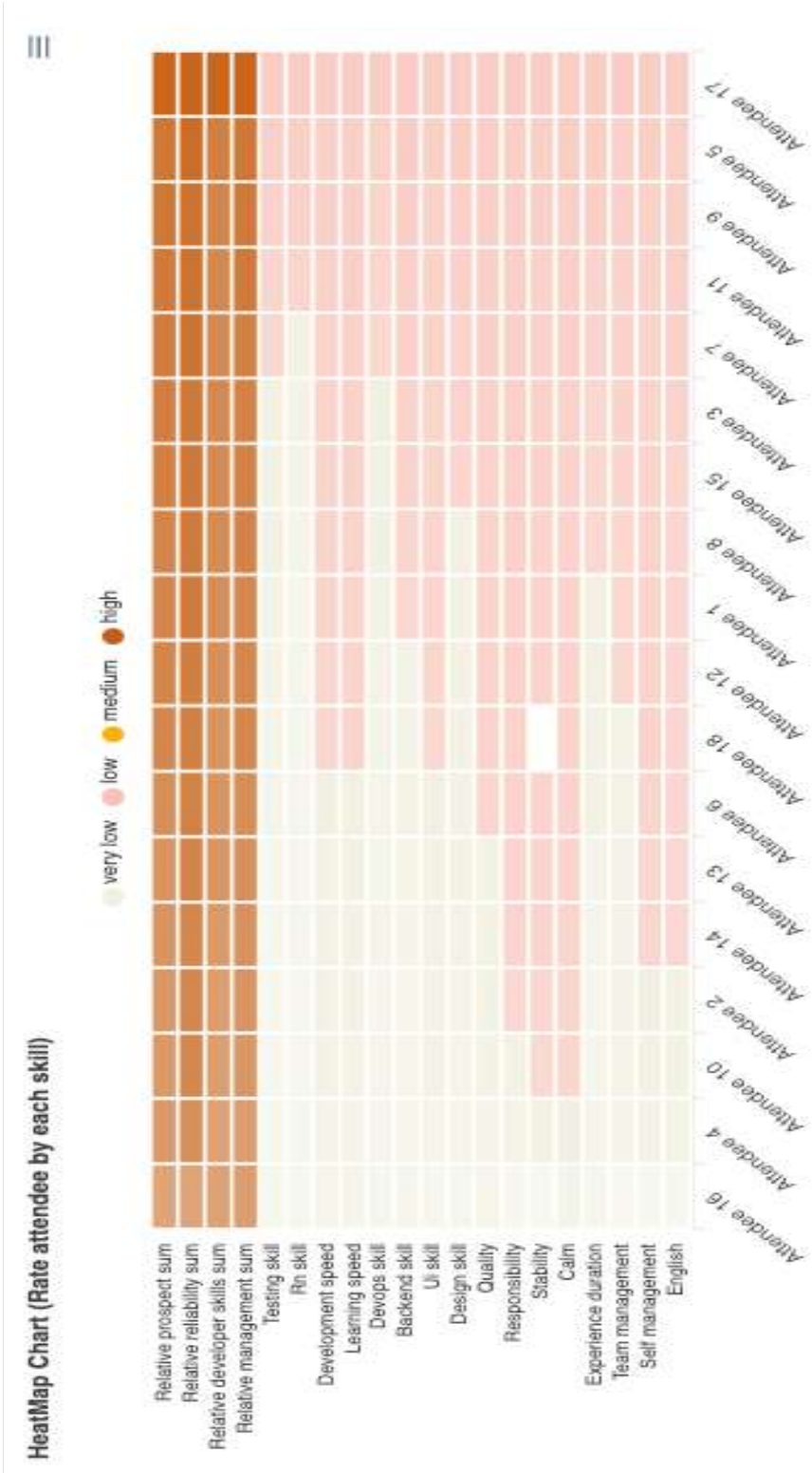


Рисунок 4.8 – Рейтинг за найбільш використовуваними пріоритетами у вигляді heat map

На рисунку 4.8 ви можете побачити рейтинг учасників і рейтинг кожного учасника для кожної навички. Високий – значення від 10 до 110, середній – значення від 7 до 10, низький – значення від 3 до 7, дуже низький – значення від 0 до 3. Як ми бачимо з графіка, топ-3 найкращих кандидати відповідно до рейтингів усіх навичок – це Учасник 9, Учасник 5 та Учасник 17. Однак ці 3 найкращих ще не є нашими найоптимальнішими кандидатами, оскільки нам потрібно скласти рейтинговий список лише для тегів, які ми отримали на малюнках (4.6 – 4.7), а також врахувати бажані та не бажані теги кожного учасника рейтингу.

На рисунку 4.9 ви можете побачити рейтинг учасників і рейтинг кожного учасника для кожної навички яка вже була відфільтрована на основі даних із попереднього алгоритму. Високий – значення від 10 до 110, середній – значення від 7 до 10, низький – значення від 3 до 7, дуже низький – значення від 0 до 3. Як ми бачимо з графіка, топ-3 найкращих кандидати відповідно до рейтингів усіх навичок – це Учасник 9, Учасник 5 та Учасник 17. Однак ці 3 найкращих ще не є нашими найоптимальнішими кандидатами.

Далі був зроблений рейтинг лише по тих тегах які були присвоєні саме задачі на попередньому етапі. Високий – значення від 10 до 15, середній – значення від 8 до 10, низький – значення від 4 до 8, дуже низький – значення від 0 до 4. Як ми бачимо з графіка, топ-3 найкращих кандидати відповідно до рейтингів усіх навичок – це Учасник 3, Учасник 14 та Учасник 1. Однак ці 3 найкращих ще не є нашими найоптимальнішими кандидатами, хоча здається що по всім оцінках Учасник 3 це наший шуканий кандидат.

Далі був створений рейтинг із впливом суми загальної оцінки часу потрібного для виконання решти задач учасників які мають статус To do або Next. Як ми бачимо з графіка, топ-3 найкращих кандидати відповідно до рейтингу – це Учасник 8, Учасник 14 та Учасник 6. Однак ці 3 кандидати хоч і мають невеликий діапазон розриву в оцінках, потрібно проаналізувати ще етапи впливу тегів preferred а також not preferred.

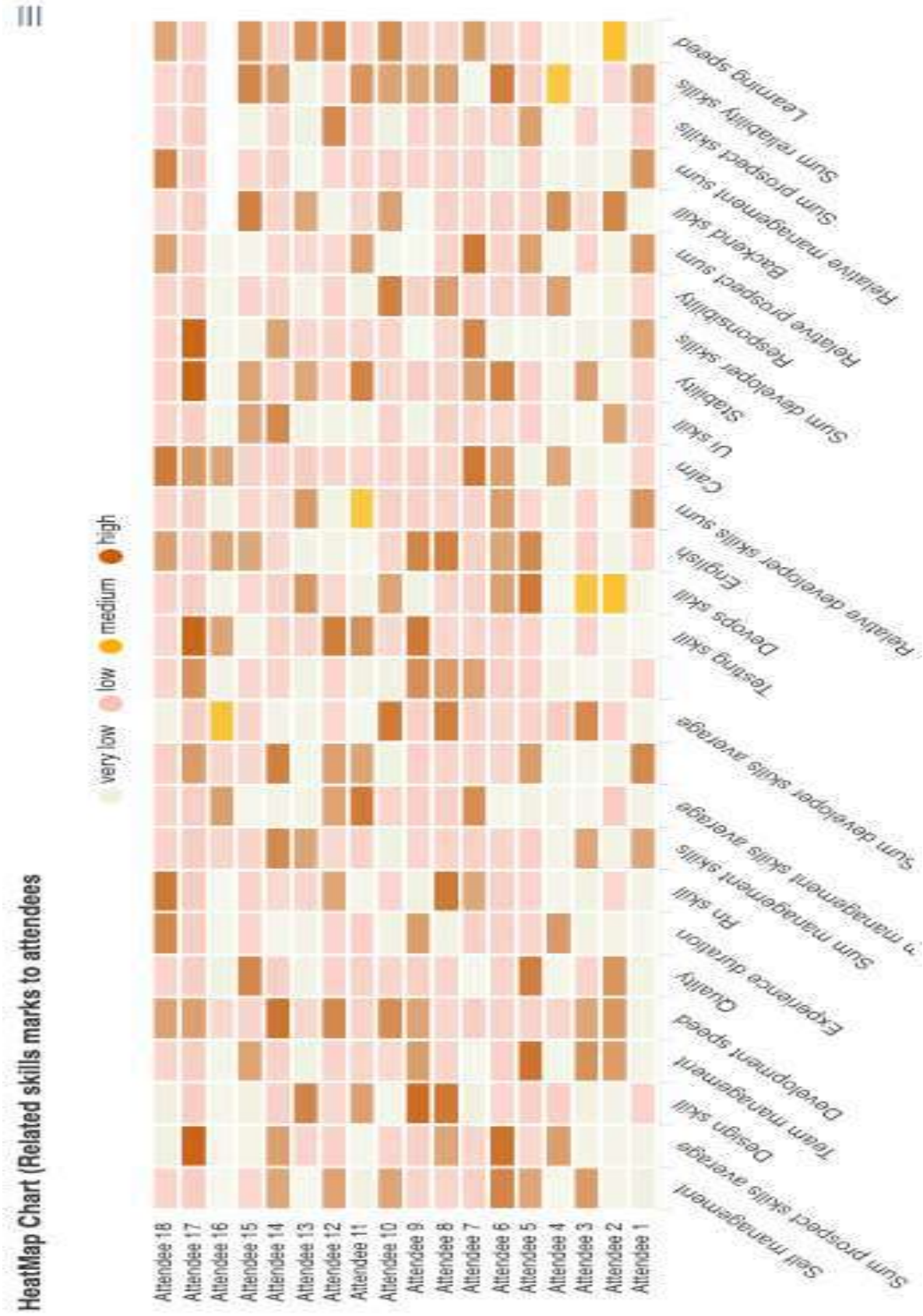


Рисунок 4.9 – Рейтинг за використовуваними тегами навичок для завдання

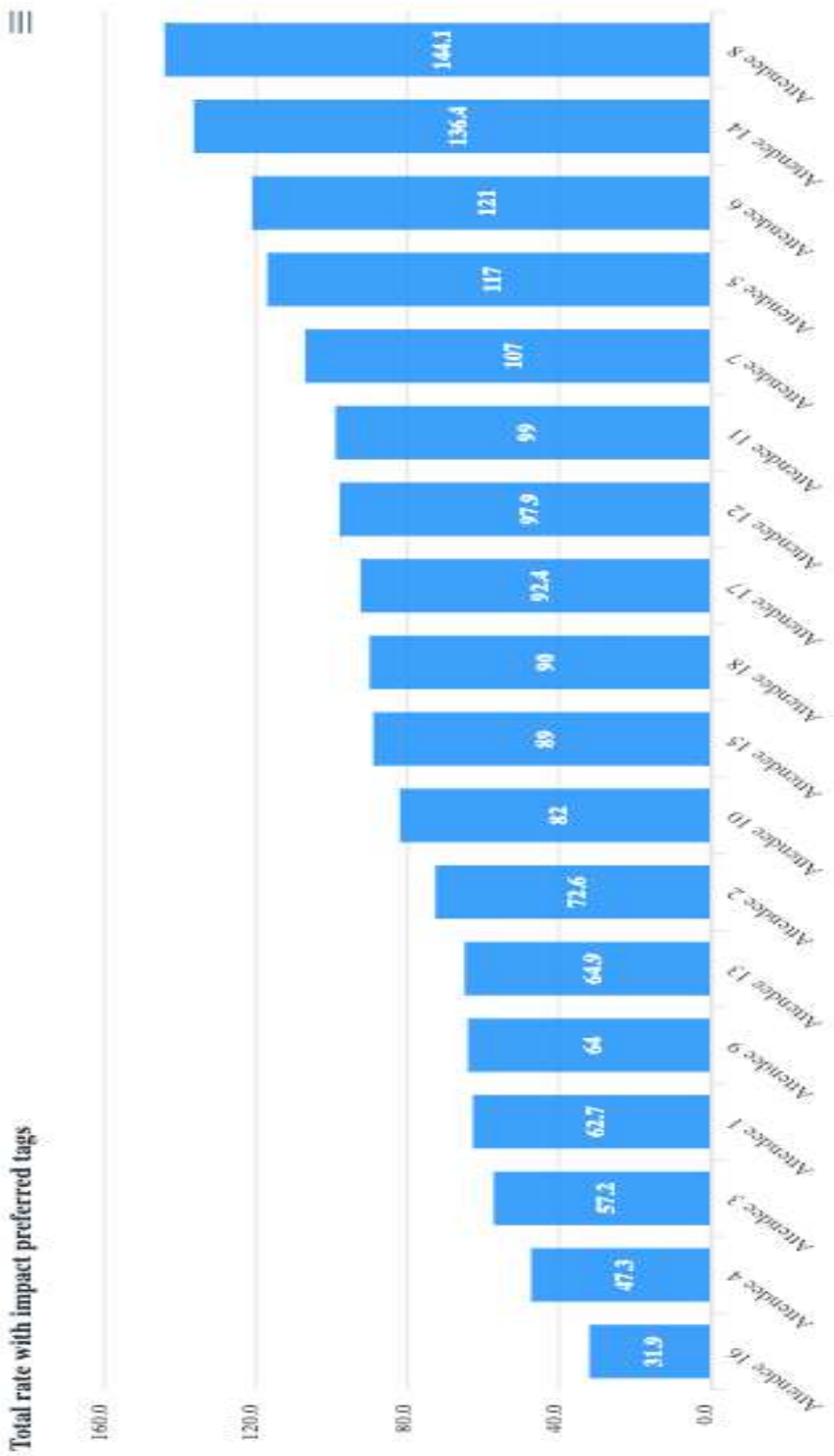


Рисунок 4.10 – Рейтинг із впливом preferred tags

Беручи до уваги теги, з якими учасники хотіли б працювати, ми додали всі бали за тегами та врахували отримані побажання, ми отримали один загальний

рейтинг, як на малюнку 4.10. З нього ми бачимо, що кандидати, які були в попередній рейтинг Учасник 9, Учасник 5 та Учасник 17 змінили свої позиції в цьому рейтингу. Тепер ТОП-3 посіли учасники Учасник 6 - 121 бал, Учасник 14 - 136,4 бал, Учасник 8 - 144,1 бал. Наступним кроком також є врахування тегів, з якими учасники не захотіли працювати у вже згасаючому рейтингу.

Беручи до уваги теги, з якими учасники не хотіли б працювати, ми змінили рейтинг на основі кількості тегів, з якими учасники не хотіли б працювати. Ми отримали один загальний рейтинг, як показано на малюнку 4.11. З нього ми бачимо, що кандидати, які були в попередньому рейтингу Учасник 6 - 121 бал Учасник 14 - 136,4 бали, і Учасник 8 - 144,1 бали змінили свою позицію в цьому єдиному Учасник 6 на Учасник 5. Зараз трійку лідерів займають учасники Учасник 5 - 117 балів Учасник 14 - 124 бали, Учасник 8 - 131 бал. Як бачимо, у всіх учасників з першої трійки бали знизилися. Це означає, що ці учасники з топ-3 мають принаймні один тег, з яким вони не хотіли б працювати. Наступним кроком є також врахування завантаженості людей за загальною оцінкою завдань і загальної кількості завдань зі статусом виконання.

Враховуючи завантаженість людей за загальною оцінкою завдань і сумарних завдань зі статусом виконання, ми змінили рейтинг і отримали один загальний рейтинг, як на малюнку 4.12. З ним ми бачимо, що кандидати, які були в попередньому рейтингу Учасник 5 – 117 балів, Учасник 14 – 124 бали, Учасник 8 – 131 бал знову змінили свою позицію Учасник 5 на Учасник 6. Тепер у першій трійці учасників Учасник 5 – 108,9 бал, Учасник 14 – 109,12 бал та Учасник 8 – 115,28 бал.

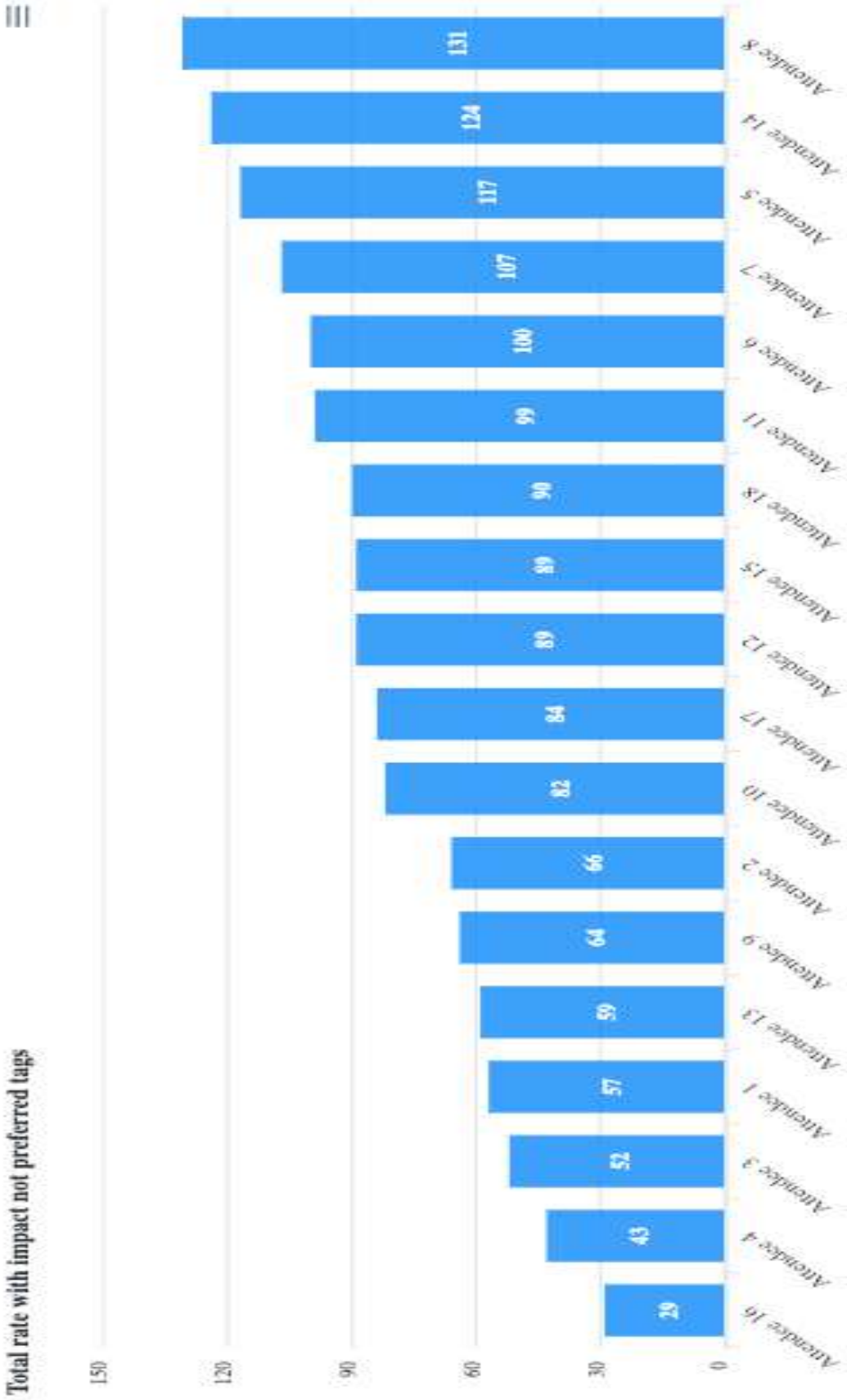


Рисунок 4.11 – Рейтинг із впливом not preferred tags

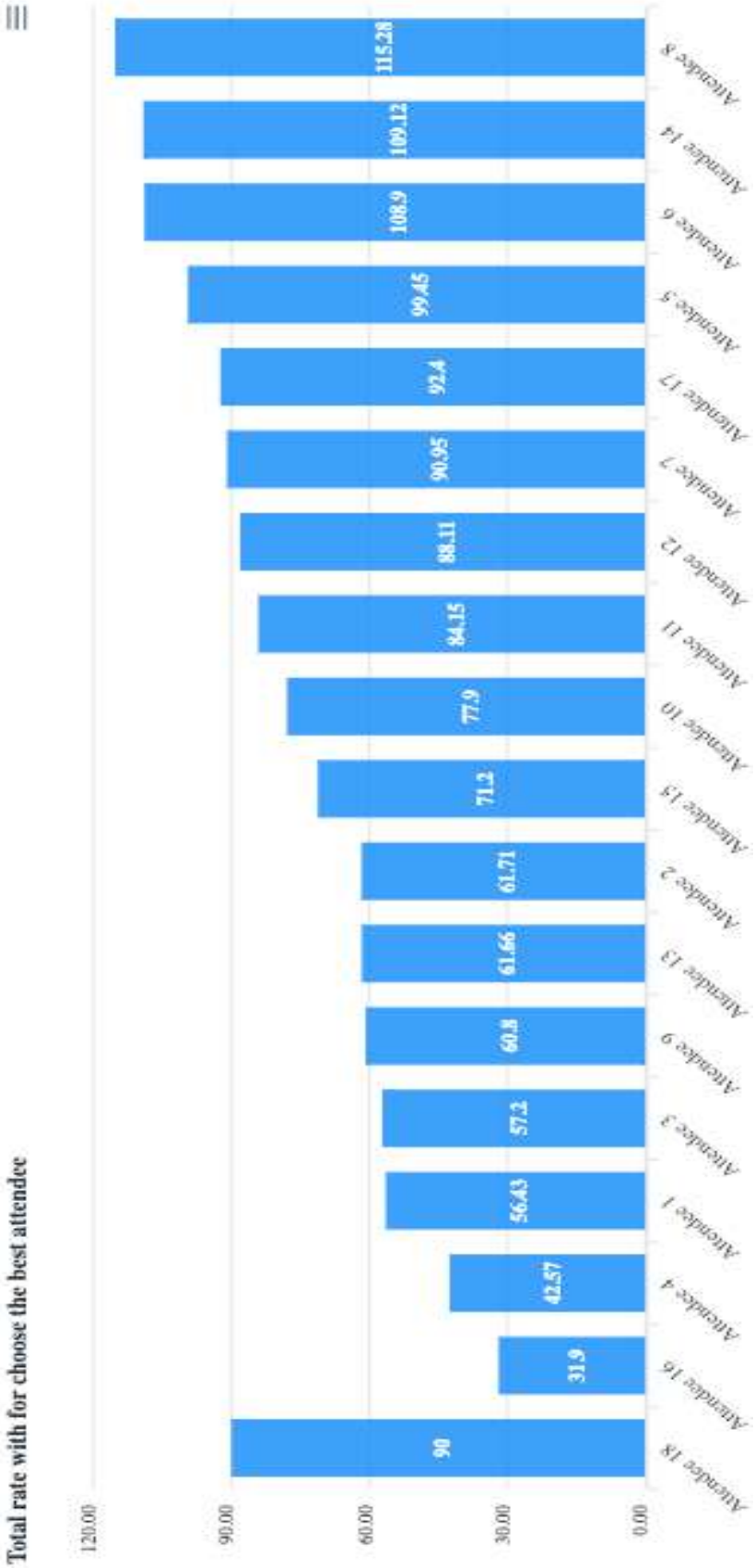


Рисунок 4.12 – Результуючий рейтинг для вибору найкращого кандидата

Як бачимо, у всіх учасників з першої трійки бали пошкоджені. Це говорить про те, що ці учасники з ТОП-3 мають певну кількість завдань, які впливають на загальний рейтинг, але не суттєво. А учасник 5 вже значною кількістю завантажень і йому не можна давати нові завдання. Оскільки немає учасників з однаковими балами, то з малюнка 8 можна сказати, що найоптимальнішим варіантом виконання завдання є Учасник 8. Оскільки він має 115 балів, то він має незначне менше значення від учасників Учасника 6 (108) та Учасника 14. (109).

Отже за результатами автоматичного позначення завдання можна побачити на малюнках 4.6 - 4.8. На малюнку 4.6 видно, що найкращий тег класифікації для пріоритету – високий. Спринт є найкращим для етапу тег групи рисунок 4.8. З великим розривом у навичках, тег react. А для обов’язкового типу маємо Завдання.

Далі за отриманими тегами почали пошук найкращого кандидата для виконання завдання.

На першому етапі пошуку був отриманий рейтинг усіх гравців за всіма тегами рисунок. 4.8. Тоді з поточного рейтингу було сформовано новий з урахуванням завантаження людей. Потім ще один із впливом бажаних тегів рисунок 4.10 і не бажаних рисунок 4.11. Потім було складено останній рейтинг для вибору найкращого виконавця завдання рисунок. 4.12. З прикладу ми бачимо, що учасник - 8 є найкращим варіантом.

4.4 Висновки

Було розроблено та реалізовано автоматизовану інформаційну систему для визначення найкращого кандидата для виконання завдання, а також для класифікації завдання та додавання до нього тегів класифікації, що забезпечує точний вибір найкращого виконавця завдання та виконує точну класифікацію завдання та визначає відповідні теги. Подальші дослідження спрямовані на підвищення точності та автономності моделі та покращення рівня класифікації до рівня підгруп тегів.

Запропонована система розподілу та оцінки завдань у процесі розробки програмного забезпечення спрямована на підвищення точності та автономності моделі, а також підвищення рівня класифікації до рівня підгруп тегів. Система використовуватиме алгоритми машинного навчання, щоб визначити найкращого кандидата для завдання, а також класифікувати завдання та додати відповідні теги. Система також зможе визначити найбільш підходящі завдання для конкретного кандидата, виходячи з його навичок і досвіду. Крім того, система зможе забезпечувати зворотний зв'язок з кандидатом щодо його ефективності, дозволяючи йому вдосконалювати свої навички та ставати більш ефективними у своїй роботі. Система також зможе відстежувати хід виконання завдання та надавати своєчасні оновлення зацікавленим сторонам. Нарешті, система зможе надавати детальні звіти про виконання завдання та кандидата, що дозволить краще приймати рішення.

Пропонована система розподілу та оцінки завдань у процесі розробки програмного забезпечення для методів Scrum та Agile розроблена для забезпечення автоматизованого та точного способу призначення завдань найкращому кандидату, а також класифікації завдання та додавання відповідних тегів. Система розроблена таким чином, щоб бути високоточною та автономною, а також забезпечувати рівень класифікації, точний аж до рівня підгруп тегів. Система також розроблена таким чином, щоб бути високоефективною та забезпечувати спрощений процес призначення завдань найкращому кандидату. Система також розроблена таким чином, щоб бути дуже гнучкою та мати можливість адаптуватися до мінливих вимог і завдань.

ВИСНОВКИ

Було розглянуто аспекти розробки системи розподілу та оцінки завдань у процесі розробки програмного забезпечення, були розроблені алгоритми які забезпечують максимально точне визначення особи, яка повинна виконати завдання, та відповідні теги класифікації завдань на основі його опису. Пропонована система забезпечує точну та швидку ідентифікацію особи та групи тегів на основі опису завдання.

Були проаналізовані проблем, пов’язані із розподілом завдань і оцінкою в розробці програмного забезпечення. До них належать необхідність точної оцінки завдання, складність забезпечення контролю якості та потреба в ефективній комунікації між розробниками.

Далі в було розглянуто сучасний стан розподілу та оцінювання завдань. Він розглядався як різноманітні інструменти та методи, доступні для розподілу й оцінки завдань, у тому числі системи відстеження завдань, системне програмне забезпечення для керування проектами та засоби автоматичного тестування. Також було досліджено різні методи, що використовуються для оцінювання завдань, наприклад експертну перевірку, перевірку коду тощо.

Було розглянуто потенціал для подальшої автоматизації та потребу в покращенні зв’язку між розробниками. Також розглядався потенціал використання штучного інтелекту для покращення розподілу та оцінки завдань. У ньому розглядаються різні запропоновані підходи на основі комп'ютерні системи штучного інтелекту, такі як обробка природної мови, машинне навчання та глибоке навчання.

У ході виконання кваліфікаційної роботи були вирішені такі задачі:

1. Реалізовано метод для автоматичної класифікації задач.
2. Реалізовано метод для автоматичного розподілення завдань між користувачами.
3. Було побудовано клієнт в якому були реалізовані можливості створення задачі. Також перегляд тегів і людей які були обрані для

виконання поставлених задач. І було реалізовано серверну і бекенд частину яка отримувала дані виконували функції із класифікації і пошуку і повертала результат.

4. Було досліджено ефективність розроблених та реалізованих алгоритмів за допомогою графіків.

Подальші дослідження спрямовані на підвищення точності та автономності системи, а також підвищення рівня класифікації до рівня підгруп тегів. Крім того, проводяться дослідження, щоб покращити здатність системи вирішувати складні завдання та забезпечити більш ефективний і спрощений процес призначення завдань найкращому кандидату.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Сударкіна С., Кліментова М., Анічкіна І. Методи та інструменти стратегічного маркетингового планування. *Вісник Національного технічного університету «Харківський політехнічний інститут» (Економічні науки)*, Харків: ХНТУ, 2019. С. 66-78.

2. Баркова К. Методологічні основи стратегічного планування підприємства. Харків : ХНЕУ, 2019. С. 41-49.

3. Кашталян І.В. Використання систем штучного інтелекту для вдосконалення засобів планування завдань. *Інтелектуальні комп'ютерні системи та мережі*: тези доп. III наук.-практ. конф. м. Тернопіль, 26 листоп. 2020. Тернопіль: ЗУНУ, 2020. С. 13-21.

4. Ільков А.В. Особливості стратегічного планування на малих підприємствах в умовах невизначеності. *Забезпечення сталого розвитку аграрного сектору економіки: проблеми, пріоритети, перспективи*: матеріали міжнар. наук.-прак. конф. м. Дніпро, 29-30 жовт, 2020 р. Дніпро, 2020. С. 72-74

5. Остапенко Р.М. *Організаційний механізм стратегічного планування сільськогосподарських підприємств*: матеріали міжнар. наук.-прак. конф. м. Полтава, 29-30 жовт, 2020 р. Полтава, 2020. 715 с.

6. Казимір В.В. Інформаційні основи побудови телекомунікаційних мереж. *Вісник Чернігівського державного технічного університету*, Чернігів: ЧДТУ, 2013. 340 с.

7. Стеклов В.К. Інформаційна система: навч. посіб. Київ : НАНУ, 1999. 240 с.

8. Стасєв Ю.В. Комп'ютерні мережі. Технології та протоколи моделювання. Харків : ХУПС, 2014. 359 с.

9. Кривуца В.Г. Управління телекомунікаціями із застосуванням сучасних технологій. Київ, 2007. 384 с.

10. Стеклов В.К. Інформаційна система та телекомунікації: навч. посіб. Київ, 2014. 792 с.

11. Кривуца В.Г. Стеклов В., Беркман Л., Костик Б.Я. Управління телекомунікаціями із застосуванням сучасних технологій: навч. посіб. Київ, 2007. 385 с.

12. Горбатій І.В. Телекомунікаційні системи та мережі. Принципи роботи, технології та протоколи. Львів: ЛПУ, 2016. 336 с.

13. Совєтов Б.Я. Моделюючі системи. Виявлення змін середовища та каналу та кластеризація. Київ. 2015. 343 с.

14. Клімаш М.М. Сучасні трансформації в архітектурах розподілених систем. Львів, 2015. 328 с.

15. Лунтовський А.О. Етапи розвитку сучасних інформаційно-телекомунікаційних послуг та енергоефективності мережевих технологій. *Вісник Національного університету «Львівська політехніка»*. Львів : Вид-во Львівської політехніки, 2014. 131-139 с.

16. Брукс Ф. Міфічний людино-місяць, або Як створюються програмні системи. Київ, 2006. 304 с.

17. Вендров А.М. Проектування програмного забезпечення економічних інформаційних систем. Запоріжжя. 2005. 544 с.

18. Ілес П. Що таке архітектура програмного забезпечення? URL: http://www.interface.ru/home.asp?artId=2116 (дата звернення 26.02.2023).

19. Коцуба І.Ю., Чунаєв А.В., Шиков А.Н. Основи проектування інформаційних систем: навч. посіб. СПб, 2015. 206 с.

20. Мінухін С.В., Бесєдовський О.М., Знахур С.В. Методи та моделі проектування на основі сучасних засобів CASE: навч. посіб. Харків : ХНЕУ, 2008. 272 с.

21. Пущин М.Н. Проектування інформаційних систем. МІЕТ, 2008. 234 с.

22. В.І. Грекул, Г.Н. Денищенко, Н.Л. Коровкіна. Проектування інформаційних систем: навч. посіб. Київ, 2005. 304 с.

23. Реінжиніринг бізнес-процесів. URL: https://library.if.ua/book/28/1899.html (дата звернення 14.03.2023).

24. Соммервіль І. Інженерія програмного забезпечення. 2002. - 624 с.

25. Фаулер М. Архітектура корпоративних програмних систем. М, 2005. 576 с.

26. Технології проектування. URL: https://studfile.net/preview/3997729/page:5 (дата звернення 17.03.2023).

27. Брауде Еге. Дж. Технологія розробки програмного забезпечення: навч. посіб. СПб: Пітер, 2004. 655 с.

28. К. К. Аггарвал. Система розподілу та оцінки завдань у процесі розробки програмного забезпечення. *International Journal of Computer Sciences and Information Technologies*. 2014. № 5, С. 441–445,

29. М. А. Хан. Системний підхід до розподілу та оцінки завдань у процесі розробки програмного забезпечення. International Journal of Computer Science and Engineering. 2016. №2. С. 1–7.

30. С. К. Шарма. Системний підхід до розподілу та оцінки задач у процесі розробки програмного забезпечення. *International Journal of Computer Science and Applications*. 2016. № 2. С. 1–4.

31. М. С. Хан. Розподіл завдань та оцінка у процесі розробки програмного забезпечення, *International Journal of Computer Applications*. 2011. №8. С. 1-5,

32. К. Гупта. Розподіл завдань та оцінка у процесі розробки програмного забезпечення. *International Journal of Computer Science and Engineering*. 2012. № 3. С. 545-550.

33. С. К. Джейн. Розподіл завдань та оцінка у процесі розробки програмного забезпечення. *International Journal of Computer Sciences and Information Technologies*. 2015. №5. С. 1-5.

34. Р. К. Шарма. Розподіл завдань та оцінка у процесі розробки програмного забезпечення. *International Journal of Computer Applications*. 2015. №5. С. 545-550.

35. В. К. Сінгх. Розподіл завдань та оцінка у процесі розробки програмного забезпечення. *International Journal of Computer Applications*. 2013. №6. С. 1-5.

36. П. К. Гупта. Розподіл завдань та оцінка у процесі розробки програмного забезпечення. *International Journal of Computer Sciences and Information Technologies*. 2016. №7, С.545-550.

37. А. К. Верма. Розподіл завдань та оцінка у процесі розробки програмного забезпечення. *International Journal of Computer Applications*. 2013. №10, С.1-5.

38. С. К. Ядав. Розподіл завдань та оцінка у процесі розробки програмного забезпечення. *International Journal of Computer Science and Technology*. 2014. №8, С. 545-550.

39. Р. К. Джейн. Розподіл завдань та оцінка у процесі розробки програмного забезпечення. *International Journal of Computer Sciences and Information Technologies*. 2017. №9, С.1-5.

40. М. С. Хан. Всебічне дослідження розподілу та оцінки завдань у процесі розробки програмного забезпечення. *International Journal of Computer Science and Network Security*. 2014. №8, С.1-7.

41. С. К. Джейн. Огляд розподілу та оцінки завдань у процесі розробки програмного забезпечення. *International Journal of Computer Sciences and Information Technologies*. 2016. №5, С.1-7.

42. М. А. Аль-Халіфа. Розподіл завдань у багатоагентних системах: всебічне дослідження. *International Journal of Computer Science and Network Security.* 2018. №4, С.1-10.

43. ДСТУ. Systems and software engineering – Software Life Cycle Processes. ISO 12207:2008. - [Чинне від 2008-02-01] - II, 122 с. (Інформація та документація).

44. Жогольов Є.А. Технологія програмування. М:Науковий світ, 2004. 216 с.

45. Іванова Г.С. Технологія програмування: навч.-метод. посіб. М.: Вид-во МДТУ ім. н.е. Баумана, 2002. 320 с.

46. Грищенко В.М. Метод об'єктно-компонентного проектування програмних систем. *Проблеми програмування*. 2007. Тернопіль. С. 113-125.

47. Алексенко. О. В. Технології програмування та створення програмних продуктів: конспект лекцій. Суми: СумДУ, 2013. С. 80-97.

48. Спіральна модель. URL: http://ua.wikipedia.org/wiki (дата звернення 01.04.2023).

49. Ендрю Траск . Грокаємо Глибоке навчання. 2019. С. 118-154.

50. Джон Сміт. Розподіл завдань і оцінка в розробці програмного забезпечення. *Journal of software engineering*. 2018 р. №3. 96 с.

51. Сара Джонсон. Гнучке управління завданнями: порівняльне дослідження. *International Journal of Agile Development*. 2019. №2. С. 12-15.

52. Девід Браун. Декомпозиція та розподіл завдань для проектів розробки програмного забезпечення. *IEEE Transactions on Software Engineering*. 2017. №3. С. 63-77.

53. Рейчел Грін. Систематичний огляд розподілу завдань у групах розробки програмного забезпечення. *Journal of Systems and Software*. 2016. С.12-17.

54. Майкл Лі. Розробка програмного забезпечення на основі завдань: практичне дослідження. *ACM Transactions on Software Engineering and Methodology*. 2015. С.23-27.

55. Дженніфер Кім. Розробка програмного забезпечення на основі завдань: емпіричне дослідження. *Empirical Software Engineering*, 2014. С.12-19.

56. Дженніфер Кім. Розподіл завдань у групах розробників програмного забезпечення: огляд літератури. *Journal of Software Engineering Research and Development*. 2013. С 45-48.

57. Карен Браун. Декомпозиція та розподіл завдань у гнучкій розробці програмного забезпечення. *Journal of Agile Development*. 2012. 112 с.

58. Лаура Джонс. Розробка програмного забезпечення на основі завдань: огляд поточних практик. *Journal of Software Engineering Research and Development*., 2011. С. 78-79.

59. Томас Вілсон. Декомпозиція та розподіл завдань у розробці розподіленого програмного забезпечення. *Journal of Distributed Computing*, 2010. 33 с.

60. Томпсон М. Розподіл завдань у розробці програмного забезпечення: порівняльне дослідження гнучких і традиційних підходів. *Journal of Software Maintenance and Evolution.* 2009. 87 с.

61. Бем Б. В. Економіка програмної інженерії. *IEEE Computer*. Нью-Джерсі: Прентіс-Хол. 1981. С. 12-18.

62. Брукс, Ф. П. Жодна срібна куля: сутність і випадковості програмної інженерії. 2004. №1. С.10-19.

63. ДеМарко Т. та Лістер Т. Peopleware: продуктивні проекти та команди. Нью-Йорк: Дорсет Хаус. 1985. 43 с.

64. Прессман, Р. С. Інженерія програмного забезпечення: підхід практики. Нью-Йорк: McGraw-Hill. 2010. 78 с.

65. Ройс В. В. Управління розробкою великих програмних систем. *IEEE WESCON*, 1-9. 1970. С. 44-54.

66. Соммервіль І. Інженерія програмного забезпечення. №9. Бостон. Массачусетс: Addison-Wesley. 2011. 112 с.

67. Сазерленд Дж. і Швабер К. Керівництво по Scrum. URL: https://www.scrum.org/resources/scrum (дата звернення 12.04.2023).

68. Прентіс-Хол, Р. Л. Гласс. Конфлікт програмного забезпечення. 1992 р. 14 с.

69. Аддісон-Уеслі, Т. Гілб. Принципи управління програмною інженерією. 1988. С. 56-59.

70. Прессман Р. Інженерія програмного забезпечення: підхід практикуючого. 1992. С. 17-18.

71. Фаулер М. Рефакторинг: покращення дизайну існуючого коду. 1999. С. 23-25.

72. Бек К. Пояснення екстремального програмування: сприймайте зміни. 1999 р. С. 17-18.

73. Кусумано М., Селбі Р. Секрети Microsoft, *Free Press*, 1995. С. 78-79.

74. ДеМарко Д., Лістер Т, Дорсет Хаус. Peopleware: Продуктивні проекти та команди. 1987. С. 112-116.

75. Дж. Хайсміт, А. Кокберн. Гнучка розробка програмного забезпечення: бізнес інновацій. 2001. С. 56-57.

76. Дж. Мартін, Прентіс Холл. Чистий кодер: Кодекс поведінки для професійних програмістів, 2011. С 118-124.

77. Джон Вайлі, Барлоу Дж. Гнучке управління проектами: створення інноваційних продуктів. 2019. С. 28-32.

78. Кокберн А. Гнучка розробка програмного забезпечення. 2000. 123 с.

79. Кон М. Застосовані історії користувачів: для гнучкої розробки програмного забезпечення. 2004. С. 71-73.

80. Де Марко Т, Лістер Т. Peopleware: продуктивні проекти та команди. Видавництво Dorset House. 1999. С.118-119.

81. Dingsøyr, T., Nerur, S., Balijepally, V., Moe, N. B. Десятиліття гнучких методологій: до пояснення гнучкої розробки програмного забезпечення. *Journal of systems and software*. 2012. №6. С. 1213-1221.

82. Хайсміт Дж. Гнучкі екосистеми розробки програмного забезпечення. 2002. С. 11-13.

83. Хамфрі В. С. Управління програмним процесом. 1989. 19 с.

84. Ларман К. Гнучка та ітеративна розробка: посібник для менеджера. *Pearson Education*. 2004. 97 с.

85. Паттон Дж. Відображення історій користувача: дізнайтеся всю історію, створіть правильний продукт. *O'Reilly Media*. 2014. 302 с.

86. Пелгрейв Макміллан. Прессман Р. С., Максим Б. Р. Інженерія програмного забезпечення: підхід практики. 2014. С. 117-123.

ДОДАТОК А
(обов'язковий)

ЛІСТИНГ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ГОЛОВНОЇ ПРОГРАМИ ПОШУКУ НАЙКРАЩОГО КАНДИДАТА ТА КЛАСИФІКАЦІЇ ЗАДАЧІ

Модуль «Пошук найкращого кандидата базуючись на описі завдання».

```
/**
 * This function helps to make auto-assign attendee to task
 *
 * @method
 * @param {Object} task full task data for update attendee in task
 * @param {string} workspaceId  id of workspace where to task was created
 * @param {Boolean} isShouldReturnAllRelatedData -  return all data if true and if
false return just total rate
 * @returns {string|boolean} - if the best attendee was found return attendee id
else false
 */
async handleAutomaticallyAssign(
  task,
  workspaceId,
  isShouldReturnAllRelatedData = false
) {
  const { tags = [] } = task

  /* Get all related data to assign process from Firestore and RDB*/
  const pollService = new PollService()
  const attendeeService = new AttendeeService()

  // Define which tags related to tag groups
  const { typeGroupTags, relatedTagsToTagGroup } =
    await this.getTypeOfSkillsByTagGroups(tags, workspaceId)

  // Get attendees documents from firestore
  const attendeesFromFirestore =
    await attendeeService.getAttendeesFromFirestore(workspaceId)

  // Get attendees ids and remove from array Trond and Oleksiy accounts
  const attendeeIds = attendeesFromFirestore
    .map(({ _id }) => _id)
    .filter(
      (attendee) => !ATTENDEES_THAT_SHOULD_NOT_BE_IN_RATE.includes(attendee)
    )

  // Get all tags(from RDB) usage related to attendee
  const attendeesTags = await attendeeService.getAttendeesTagsFromRDB(
    attendeeIds,
    tags,
    workspaceId
  )

  // Get all polls for each attendee from firestore
  const polls = await pollService.getPolls(attendeeIds, workspaceId)

  // Filter polls which take part in the rating assignment
  const pollsForRate = pollService.filterPollForCreatingRate(
    attendeeIds,
    polls
```

```
)

if (Boolean(pollsForRate?.length) && Boolean(attendeeIds?.length)) {
  // Transform all data before create rate
  const attendeeAndRelatedData =
    await this.transformAttendeeDataForCreatingRate({
      pollsForRate,
      attendeesFromFirestore,
      typeGroupTags,
      relatedTagsToTagGroup,
      attendeesTags,
      workspaceId
    })

  // Crate rate system without calculating total by attendees
  const rateWithoutTotal = this.createRate(
    attendeeAndRelatedData,
    typeGroupTags
  )

  // Transform data before calculate total mark by attendees
  const transformedRateWithTotal =
    this.transformDataBeforeCalculateTotal(rateWithoutTotal)

  // Sum all marks in rest rates and create total rate
  const totalRate = this.calculateTotal(
    transformedRateWithTotal,
    attendeeIds
  )

  //  Check if attendees have preferred tags, and update total marks
  const totalRateWithInfluenceOfPreferredTags =
    this.updateTotalMarkAttendeesBasicOnPreferredAndNotPreferredTags({
      totalRate,
      attendeeAndRelatedData,
      tags,
      isPreferredTags: true
    })

  // Check if attendees have not preferred tags, and update total marks
  const totalRateWithInfluenceOfNotPreferredTags =
    this.updateTotalMarkAttendeesBasicOnPreferredAndNotPreferredTags({
      totalRate,
      attendeeAndRelatedData,
      tags,
      isPreferredTags: false
    })

  // Transform total rate with impact of preferred and not preferred tags
  const rateForChooseAttendee = this.transformRateForChoosingAttendee({
    totalRate,
    totalRateWithInfluenceOfPreferredTags,
    totalRateWithInfluenceOfNotPreferredTags
  })

  const decreasedAttendeeMarksBasicOnEstimationTime =
    this.decreaseMarkOfAttendeeInRateBasicOnTheirEstimationForToDoTasks(
      rateForChooseAttendee,
      attendeeAndRelatedData
    )

  // Transform data before choose attendee
  const transformDataBeforeChooseAttendee =
```

```
      this.checkIfKeysInObjectHaveSameMarks(
        decreasedAttendeeMarksBasicOnEstimationTime
      )

    // Choose attendee
    const chosenAttendee = this.chooseAttendee(
      transformDataBeforeChooseAttendee,
      attendeeAndRelatedData
    )

    if (isShouldReturnAllRelatedData) {
      return {
        chosenAttendee,
        attendeesTags,
        totalRate,
        totalRateWithInfluenceOfPreferredTags,
        totalRateWithInfluenceOfNotPreferredTags,
        rateForChooseAttendee,
        decreasedAttendeeMarksBasicOnEstimationTime,
        transformDataBeforeChooseAttendee
      }
    }

    return chosenAttendee
  } else {
    return false
  }
}
```

Модуль «Класифікації завдання базуючись на описі завдання».

```
/**
 * This function helps to handle automatically add tags to task
 *
 * @method
 * @param {Object} task  - task from Firestore
 * @param {Boolean} isShouldReturnAllRelatedData -  return all data if true and if
false return just total rate
 * @returns {Promise} string[] - Array of new tags
 */
async handleAutomaticallyAddTagToTask(
  task,
  isShouldReturnAllRelatedData = false
) {
  try {
    if (!task) {
      console.log('task in handleAutomaticallyAddTagToTask is required')
      return
    }
    const { title, tags } = task

    const mandatorySetOfTagsThatNotWasDefined =
      this.checkIfTaskHasAllMandatoryTags(tags)

    if (
      Boolean(mandatorySetOfTagsThatNotWasDefined?.length) &&
      task._createdBy !== 'API'
    ) {
      const dataSetService = new DatasetService(this.workspaceId)

      const keywords = dataSetService.getKeywordsFromTask(title)
```

```
      const tagsDataset = await dataSetService.getAllTagsByKeywords(keywords)

      const mostUsedTagsForEachMandatoryTag =
        this.getMostUsedTagsForEachMandatoryTag(
          tagsDataset,
          mandatorySetOfTagsThatNotWasDefined
        )

      const suitableTags = this.chooseMostUsedTags(
        mostUsedTagsForEachMandatoryTag
      )

      const onlyTagNames = suitableTags
        .map((mandatoryTag) => Object.values(mandatoryTag).flat())
        .flat()

      if (isShouldReturnAllRelatedData) {
        return {
          suitableTags: onlyTagNames || [],
          mostUsedTagsForEachMandatoryTag
        }
      }

      return onlyTagNames || []
    } else return []
  } catch (error) {
    console.error('Error during handle automatically add tags to task', error)
  }
}
```

ДОДАТОК Б
(обов'язковий)

КОПІЯ НАУКОВОЇ СТАТТІ

УДК 004.4

D. V. OKRUSHKO, A. S. KASHTALIAN
Khmelnytsky National University, Khmelnytsky, Ukraine

SYSTEM OF DISTRIBUTION AND EVALUATION OF TASKS IN THE SOFTWARE DEVELOPMENT PROCESS

The paper is devoted to the theme of system of task distribution and evaluation in software development. Applied aspects of development of a system of allocation and evaluation tasks in the process of developing software for further analysis, which ensures the most accurate determination of the person who must perform the task and the appropriate classification tags for the tasks based on its description, are considered. The proposed system provides accurate and fast identification of a person and a group of tags based on the description of the task. The main purpose of this annotation is to provide an overview of the current state of the art in this field, as well as to discuss the advantages and disadvantages of the existing approaches.

The paper begins with a brief overview of the software development process and the role of task distribution and evaluation in it. It then describes the main approaches to task distribution and evaluation, including manual, automated, and hybrid approaches. The advantages and disadvantages of each approach are discussed in detail.

The paper then moves on to discuss the challenges associated with task distribution and evaluation in software development. These include the need for accurate task estimation, the difficulty of ensuring quality control, and the need for effective communication between developers.

The paper then examines the current state of the art in task distribution and evaluation. It looks at the various tools and techniques available for task distribution and evaluation, including task tracking systems, project management software, and automated testing tools. It also looks at the various methods used to evaluate tasks, such as peer review, code review, and automated testing.

Finally, the annotation concludes with a discussion of the future of task distribution and evaluation in software development. It looks at the potential for further automation and the need for improved communication between developers. It also looks at the potential for using artificial intelligence to improve task distribution and evaluation.

Overall, this paper provides an overview of the current state of the art in task distribution and evaluation in software development. It discusses the advantages and disadvantages of the existing approaches, as well as the challenges associated with task distribution and evaluation. It also examines the current state of the art in task distribution and evaluation, and looks at the potential for further automation and improved communication between developers. The annotation then examines the various techniques used to ensure quality control in task distribution and evaluation. These include the use of code reviews, peer reviews, and automated testing. It also looks at the various methods used to measure the effectiveness of task distribution and evaluation, such as time tracking, task completion rate, and defect rate. The annotation then looks at the potential for using artificial intelligence to improve task distribution and evaluation. It examines the various AI-based approaches that have been proposed, such as natural language processing, machine learning, and deep learning. It also looks at the potential for using AI to automate the task distribution and evaluation process.

Finally, the paper looks at the potential for using blockchain technology to improve task distribution and evaluation. It examines the various blockchain-based approaches that have been proposed, such as smart contracts and distributed ledgers. It also looks at the potential for using blockchain to improve the accuracy and speed of task distribution and evaluation.

Keywords: task planning, task distribution, task classification, frequency characteristics, search for the most optimal criterion for determining the best candidate.

Introduction. Problem Setting

Today, developing software for various computer systems is a complex process that requires careful planning, adherence to the plan, and execution. To ensure successful completion of all these tasks, it is very important to have an effective method, a system that will be responsible for distributing these tasks among the performers and evaluating these tasks. After researching some works [1] which presented many tools, information and methods for task distribution and evaluation in the process of software development and computer systems, it was found that the described methods and algorithms have their peculiarities, which do not allow to automate the system of task distribution and classification, so these approaches should be optimized and improved as much as possible, since automation of these processes will affect the coordination in the team, will give the opportunity to determine the exact deadlines for the performance of certain tasks. During the failure of certain nodes of the system, the process of forecasting, distribution and classification of tasks and all elements of the system that were not completed should be transferred to the endpoints or nodes for re-execution. The above-described process greatly affects the speed of the

entire system [2] as a whole and is a bad tone for the end user, although the mechanisms considered often have this concept as a basis, it is possible to improve the mechanisms and methods in different ways.

The purpose of the research. Development of a full-fledged autonomous automated information system for determining the best candidate for task execution, as well as task classification and adding to it pre-defined tags or tag groups.

Object of research. The process of creating an information system with ratings of the best candidates based on their practical skills assessments and classification of the task description using data used in previous learning iterations.

Subject of research. Method and information system for determining the best candidate for task execution and model or method for task classification according to tags.

Research methods. To determine the best candidate for task execution, there should be a method that provides for the creation of a rating system based on ratings given by other users, taking into account the load of a person with existing tasks, as well as taking into account the classification of the task for the presence of tags that a person gives preference to and wants to work with. For determining the classification, there will be a method that provides for the search of a group of tags that belong to a particular task and updating the task structure with new weights for classification tags.

Scientific novelty of results. Creation and modernization of algorithms for creating a new system that will work faster, more accurately and will use fewer machine resources.

Practical significance of the obtained results. Creation of a system with some number of independent APIs that will use cloud technologies and a database to store results, track the accuracy of forecasts and provide fast access to resources and their protection.

1. Analysis of existing tools for task distribution and classification.

Waterfall is a model used during project development. It was one of the first models used in the process of creating various software. The Waterfall model provides developers and managers with a step-by-step plan of work, where before moving on to the next step, all previous ones must be completed, as they are blocking for the next steps. This is very similar to the visualization in Canban, where managers and developer teams can monitor the process of existing and completed tasks with the possibility of monitoring in real time which tasks block the next stage and which tasks are more prioritized. Waterfall is implemented as a model of a sequential life cycle. Waterfall starts with the collection of the team and documentation of all necessary requirements. Then the solution design is carried out, which will solve the set tasks. Then it is all tested, covered with tests and given to the customer. But it is important to note that developers must complete the step before they start a new one.

The Waterfall model has six phases, which are shown in Figure 1.1. Where each phase was developed in such a way that it was clear when you can start the next step. During the first phase, the team only receives all the requirements from the client, makes basic documentation. They must be thoroughly studied taking into account clearly defined final goals.

On the second phase, the software design team creates all the necessary basic architecture. After determining all the necessary basic structure and necessary software, developers begin the main phase, that is, writing code.

After the basic structure is created, the necessary services are determined and access to third-party services is obtained, developers begin the main phase of implementation of the code, where each module is developed separately, which is then necessarily tested and the entire code is covered with various tests. Also, each module must be fully operational before the next phase.

During the testing and integration phase, all created modules are combined into one system, which will then become the final product. Developers or a special testing team must test the entire system and fix all the errors found.

On the next delivery and deployment phase, developers must install the program or transform it into a state that can be used by end users. Usually during this phase, automatic tests are deployed on servers. And only after the above points, the system falls into the so-called state that is used by end users.

The last phase is the launch of the software, where the developers' task is to support the product or incorporate new customer requests as well as fix any bugs that were not detected during testing.

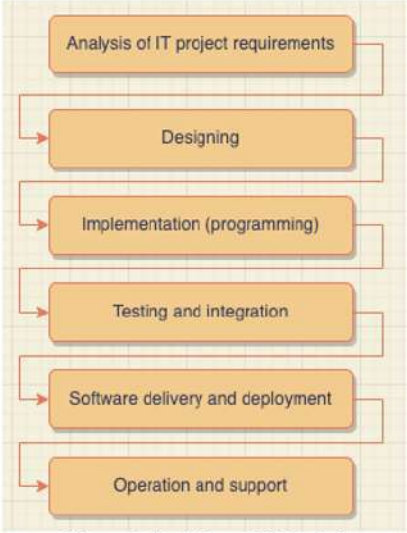


Figure 1.1 - Waterfall Model

The waterfall model is very simple to use which gives it a number of advantages such as:

1. Clearly defined phases of development.
2. Offers a large amount of documentation. All stages must be documented.
3. Teams know the amount of work that needs to be done.
4. Presence of clients only at the initial and final stages.

Over time, it became clear that the model also has its drawbacks:

1. Since the requirements may not always be clear, there is not enough flexibility to change the final result in an easy way.
2. Strict division into development phases.
3. These two simple constraints led to failures during development and task distribution, and the reasons were:
4. Presence of unrealistic project goals.
5. Inaccurate project estimates.
6. Presence of risks that are difficult to control.
7. Presence of not precisely defined requirements.
8. Lack of normal reporting on the development status.
9. Lack of communication with the customer, which affects the development process of the project in a negative way and increases the development time when it is necessary to add or correct changes.
10. Usually the use of old or inappropriate technologies for writing code.
11. Also, it may be due to excessive complexity of the project.
12. Presence of commercial pressure.
13. Lack of project development practices.

In 2001, the Agile Alliance was formed to create teams of the most flexible and dynamic methods to optimize the software development process. A document was also created that defined the basic concepts of the Agile approach and was called the Agile Software Development (Agile Alliance) [4-7]. Four main concepts were created that defined the basic values of Agile, and these values should be used in all Agile technologies (Figure 1.2).

The new values and elements contrast sharply compared to the traditional approach, which in turn relied on clearly defined plans. Following the basic principles will help increase the likelihood of success in creating the final product for wide user use.

The differences between the two models that were analyzed and studied are given in Table 1.1.

As practice shows, the total number of failed projects, or projects that are rejected, is quite large. In the work [8, 9] it was reported that only almost 40% of all existing projects are successful. About 45% were rejected for various reasons (there are no all planned functions and capabilities of software, the allocated budget was exceeded, tasks were completed untimely, etc.). And 15% were completely unsuccessful (tasks were completed, but the software did not reach the market, or the order for software was canceled). It was also noted that since 2004 there has been a gradual increase in all successful projects from 30% on average by 5%.

It was also said in the report that the size of the development project has a great influence and this is more important than the chosen methodology regardless of whether it is fast and flexible or traditional.

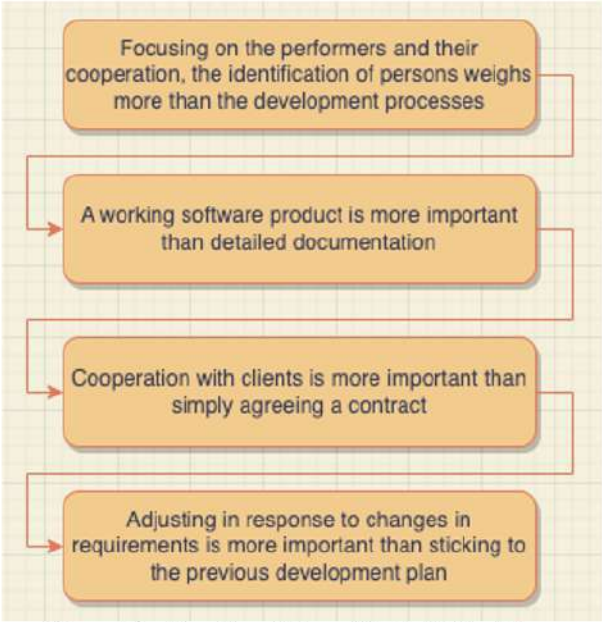


Figure 1.2 - Four Core Values of Agile Methodology

In the aforementioned work it was stated that a large project is much more likely to encounter problems (10 times) than a small project, and for small projects, flexible methods and approaches greatly simplify the work.

Today, the Agile methodology is expanding in use. As research [10, 11] shows, almost 45% of users who use flexible software development methods use Agile in almost all of their projects. A large percentage of organizations that participated in the survey (90%) say that they use flexible software development methods in all of their projects, as this increases the chances of successful project completion.

The Scrum approach (considered the best method) is used by 55% of Agile teams of managers and developers. If we take into account various modifications of the Scrum methodology, the overall usage rate increases to 73% [12].

Table 1.1 - Comparison of two Agile and Waterfall methods.

Agile	Waterfall
Short-term planning with the possibility of dynamic supplementation of necessary requirements.	Planning takes a lot of time since this process requires long-term planning, and requirements cannot be changed during the development of the software according to the approved execution plan and existing infrastructure within the system.
All team members will play an equal role in the team.	All team members follow the established hierarchical structure within the organization.
Constant communication with the client is carried out at all stages of development, which in turn allows to receive timely feedback from the client and understand whether everything was done correctly and clearly.	Completely independent from communication with the customer.
Thanks to the fact that continuous testing is carried out, there is a constant control of the quality of the software.	Late planning can significantly lag behind quality control, as the project is only tested at the last stages of the model and testing takes a lot of time. Also, bug fixes must be structured and present in the system.
It is convenient to measure and track the progress of the product due to measurements in sprints or units.	The entire progress is measured by the steps implemented
The deadlines for software implementation can be conveniently changed if additional requirements for the final product appear during one of the sprints.	Deadlines can be easily determined, since no new changes are expected and if there are no unexpected system failures.
Depending on the projects, they can be divided into different categories such as complex, current or long-term, etc.	The best approach for projects that are not of large size, and all steps can be planned or they are predictable and can be influenced. It is necessary to know all the requirements.

In the IT sphere, there is a widespread belief that teams using Agile in their approach significantly increase the chances of successful software development. Although this is not always confirmed by empirical data, Agile technology has long been seen as a methodology that will make a big breakthrough in software development for projects of various scales. Recent research in [14, 15, 16] studying the success of Agile projects shows that the possibility of such a belief can be proven on scientific data. The works showed a quantitative study to check whether the use of flexible Agile methods during software development affects the success rate of products. They noticed signs that the use of flexible planning methods in the software development process has a significantly higher registered success rate of products. Evaluation and display of results were conducted for three categories of success: overall project success, stakeholder success, and effectiveness.

Agile projects are well known for their flexible software development process compared to traditional projects. The process itself consists of a large set of practices that in turn describes a set of procedures that teams of developers then use to achieve the project's final goals.

For IT projects with the goal of creating quality software, the agile approach helps to determine the scope and size of the project at all stages, including the final stage when the project is in constant use by users. Size, required time, as well as cost and quality are quite important criteria when considering the success of a software product.

2. Development of an algorithm that will allow automation of the task distribution process.

In the begin, it is necessary to form a corresponding list of skills for each team member (Figure 1.3).

Basic skills	Basic skills
English	E(m)= E / 1.6
Self management	SM(m)= SM / 1.6
Team management	TM(m)= TM / 1.6
Experience duration	ED(m)= ED / 1.6
Calm	Ca(m) = Ca / 1.6
Stability	St(m) = St / 1.6
Responsibility	Re(m) = Re / 1.6
Quality	Qu(m) = Qu / 1.6
Design skills	DES(m) = DES / 1.6
UI skills	UIS(m) = UIS / 1.6
Backend skills	BS(m) = BS / 1.6
Devops skills	DOS(m) = DOS / 1.6
React native skill	RNS(m) = RNS / 1.6
Testing skill	TS(m) = TS / 1.6
Learning speed	LS(m) = LS / 1.6
Development speed	DS(m) = DS / 1.6

Figure 1.3 - A list of necessary skills describing a person

The maximum total score of all skills will be 160 points, since there are 16 skills in total. To find out how much each skill is worth in percentage, we divide each skill by 1.6 and get 6.25%. - S_{max}. This is the maximum value that a skill can get.

Next step, we will divide these skills into 4 main groups. These are Skill (S), Management(M), Reliability(R), Prospects(P). Let's calculate the sum of all skills that belong to the Skill group.

$$S = \sum \begin{pmatrix} DES(m), UIS(m), BS(m), DOS(m), RNS(m) \\ , TS(m) \end{pmatrix},$$

where m - this is the maximum grade in percentage. Let's now calculate the average and relative value of each skill for this group.

$$S(a) = S/6, \quad S(r) = \frac{S}{0{,}375},$$

where 0,375 is –

$$(S_{max} * 6) / 100.$$

Now let's calculate the corresponding ratings for Management (M), Reliability (R), and Prospects (P) using the same formulas.

$$M = \sum\{E(m), SM(m), TM(m), ED(m), S_a\},$$

Where S_a – this is a previously calculated value, i.e. the average value for Skill, and m is the maximum rating in percentage.

$$R = \sum\{Ca(m), SM(m), TM(m), ED(m), Re(m)\},$$

Where m - this is the maximum grade in percentage.

$$D = \sum\{LS(m), DS(m), S\},$$

where m – This is the maximum score in percentage ratio, and S is the sum of all skills of the Skill group.
And accordingly, the average values.:

$$M(a) = M/5,$$

$$R(a) = B/4,$$

$$D(a) = D/3$$

And relative meanings:

$$M(r) = \frac{M(a)}{31{,}25},$$

Where 31,25 is –

$$(S_{max} * 5)/100,$$

$$R(r) = \frac{R(a)}{0.25},$$

where 0,25 is –

$$(S_{max} * 4)/100,$$

$$D(r) = \frac{R(a)}{n},$$

Where n is –

$$D(r)\frac{(S_{max} * 2 + S(r))}{100}.$$

For the tag determination system, a data set needs to be created. This data set is formed from a graph where each vertex is connected to another vertex, i.e., a many-to-many relationship.

We will identify 5 main tag groups. These will be: Priorities, Projects, Skills, Types, Phase. Besides these main groups, there may be others. Each such tag group can contain a subset of size N, where N is a natural number. To determine the tags that are related to the task, we will use the formula:

$$Tags\{N\} = \sum_{i=1}^{N}(\overrightarrow{max}(Priotities\{SubPriorities(i)\}) + \sum_{i=1}^{N}(\overrightarrow{max}(Projects\{SubProjects(i)\}) + \sum_{i=1}^{N}(\overrightarrow{max}(Phases\{SubPhases(i)\}) + \sum_{i=1}^{N}(\overrightarrow{max}(Skills\{SubSkills(i)\}) + \sum_{i=1}^{N}(\overrightarrow{max}(Types\{SubTypes(i)\})$$

,

where SubPriorities $\in$ Priorities, Sub Projects $\in$Projects, SubPhases $\in$Phases, SubSkills $\in$Skills.

To determine who will be responsible for completing the task, a rating system needs to be created. The ratings include: the number of tasks completed by tag group, the number of tasks completed by subgroups, the number of tasks planned to be completed, the total sum of estimations for all planned tasks, the total sum of ratings, S(r) value, M(r) value, R(r) value, D(r) value, and the sum of all other skills. After creating the corresponding ratings with positions in the sub-ratings, we form one general rating. Then, taking into account the preferred and not preferred tags of each participant, we lower or raise his position in the rating.

To calculate the impact of preferred and not preferred tags, we use the following formulas:

$$preferred = \prod_i 10 * \{P_i\}/100,$$

$$not\ preferred = \prod_i 10 * \{P_i\}/100,$$

Where - P(i) is a set of preferred and not preferred tags for a task.
The formula for the influence on the overall rating in the ranking is:

$$mark = total\ mark + (preferred - not\ preferred)$$

To influence the general time for task completion, simply multiply the rating by the corresponding value in the time range {2,5,10,15,20}. The next step is to find the maximum value in the rating and in this way we find out who our potential person is best suited for the task.

4. Results of distribution and evaluation of tasks

For the experimental research, a dataset generated by us was used. To generate the dataset, a graph structure was taken as a basis, where each node has a connection with all other nodes (Figure 1.4). For a simple visualization of the prediction of the optimal candidate, let's take as an example the description of the task - "Me as a super admin, should be able to create a form for inviting users", and no qualification tags will be added. The results of auto-tagging the task can be seen in Figures 1.5, 1.6. In Figure 1.5, it can be seen that the best classification tag for priority is High. Sprint is the best for phase (Figure 1.6). With a big gap in Skills, the tag is React. And for the mandatory Type we have Task.

Next, the obtained tags need to be used to search for the best candidate to perform the task.

At the first stage, you can see the rating of all players for all tags (Figure 1.7). Then from the current rating it is necessary to form a new one taking into account the load of people. Then another one with the influence of preferred tags (Figure 1.8) and not preferred (Figure 1.9). Then build the last rating to select the best performer of the task (Figure 1.10). From the example we see that Attendee - 8 is the best option.

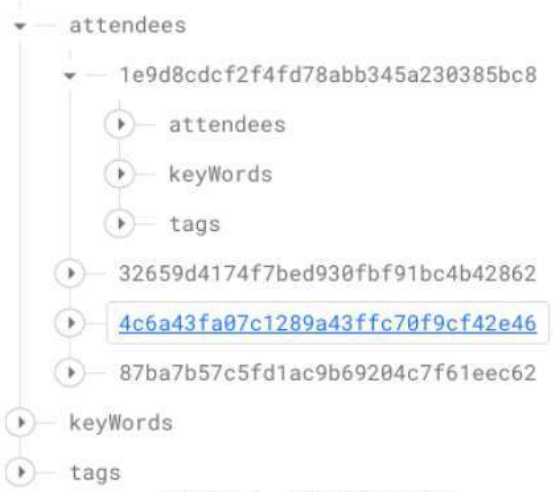


Figure 1.4 – Dataset structure

In Figure 1.4 you can see a clearly structured entity where each key has an attendee Id which is hashed. In turn, each attendee has a link to each attendee, including itself. This includes the key words which are also hashed, of the task described and all existing and used tags in the system. This will help in the future when we generate the most compatible and appropriate tags for task description.

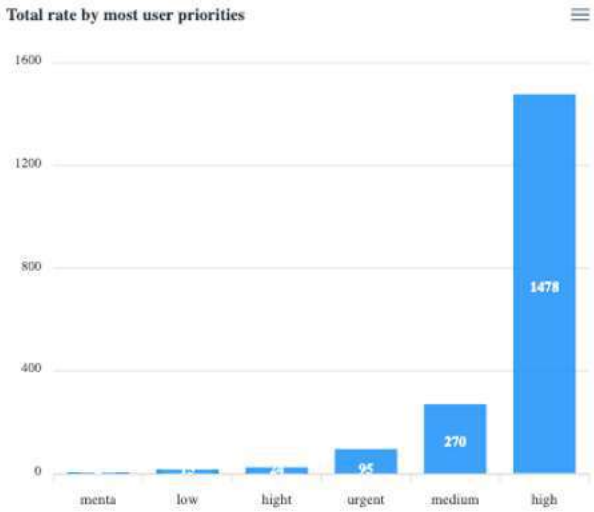


Figure 1.5 – Total rate by most used priorities

In the Figure 1.5 it can be seen that out of all the tags present in the graph structure (height, low, urgent, medium), the tag 'high' is the most frequently used for all the words described in the task "Me as a super admin, should be able to create a form for inviting users". This tag was used 1478 times. It has a huge gap from the tag 'medium' which has only 270. This suggests that the tag 'high' is probably the most optimal option for the described task.

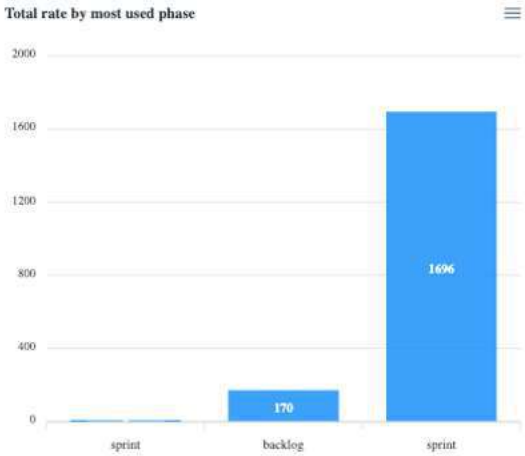


Figure 16 – Total rate by most used phase

In the Figure 1.6 it can be seen that out of all the tags present in the graph structure (sprint, backlog), the tag 'sprint is the most frequently used for all the words described in the task "Me as a super admin, should be able to create a form for inviting users". This tag was used 1696 times. It has a huge gap from the tag 'backlog' which has only 170. This suggests that the tag 'sprint' is probably the most optimal option for the described task.

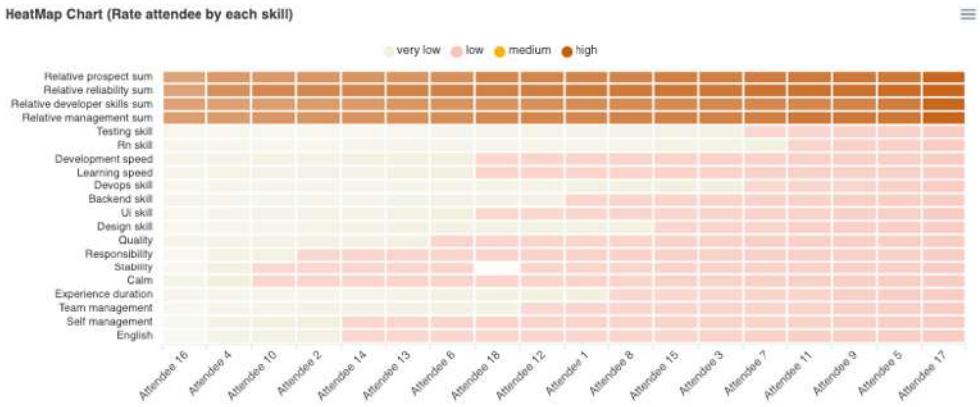


Figure 1.7 – Total heatmap of attendees by all tags

In the Figure 1.7, you can see the rating of the attendees and the rating of each attendee for each skill. High is the value from 10 to 110, medium is the value from 7 to 10, low is the value from 3 to 7, and very low is the value from 0 to 3. As we can see from the graph, the top 3 best candidates according to the ratings of all skills are Attendee 9, Attendee 5, and Attendee 17. However, these top 3 are not our most optimal candidates yet, as we need to make a rating list only for the tags we got in Figures (3-4) and also take into account the preferred and not preferred tags of each participant in the rating.

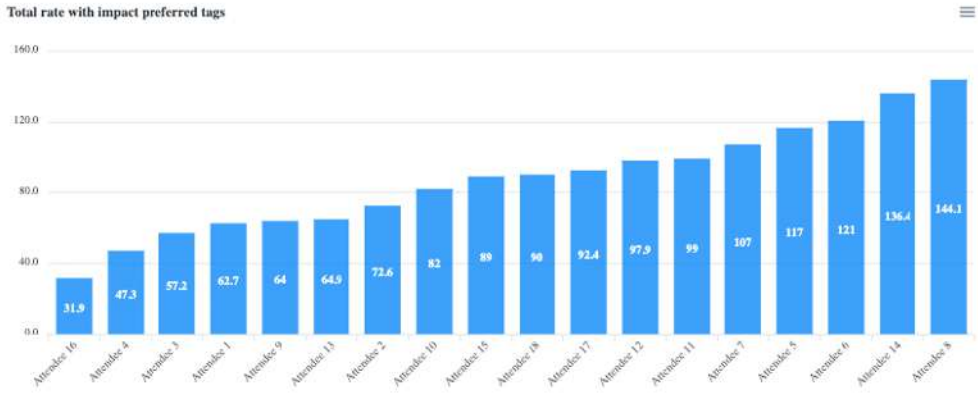


Figure 1.8 – Total rate with impact preferred tags

Taking into account the tags that participants would not like to work with, we added all the scores by tags and took into account the wishes we received, we got one overall rating as in Figure 6. From it we see that the candidates who were in the previous rating Attendee 9, Attendee 5, and Attendee 17 changed their position in this rating. Now the top 3 are taken by participants Attendee 6 - 121 points, Attendee 14 - 136.4 points, and Attendee 8 - 144.1 points. The next step is also to take into account the tags that participants did not want to work with in the already fading rating.

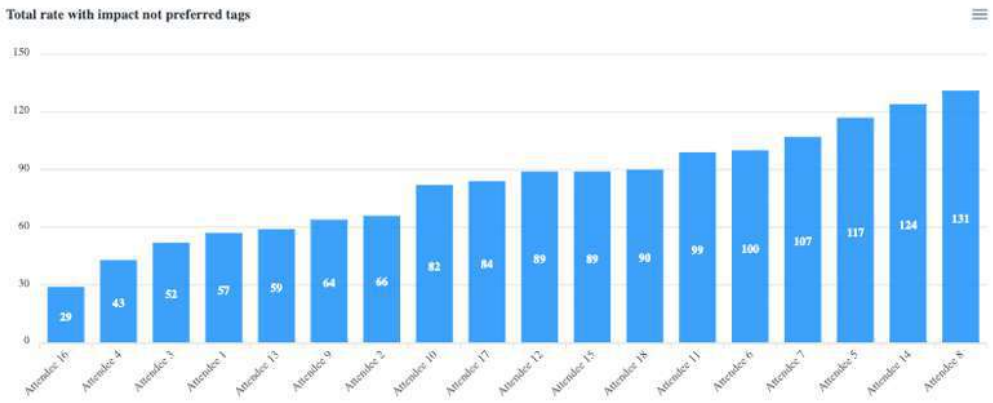


Figure 1.9 – Total rate with impact not preferred tags

Taking into account the tags that participants would not want to work with, we changed the ranking based on the number of tags that participants would not want to work with. We got one overall ranking as shown in Figure 7. From it we see that the candidates who were in the previous ranking Attendee 6 - 121 points Attendee 14 - 136.4 points, and Attendee 8 - 144.1 points changed their position in this only Attendee 6 to Attendee 5. Now the top 3 are occupied by participants Attendee 5 - 117 points Attendee 14 - 124 points, and Attendee 8 - 131 points. As we see, the scores of all participants from the top 3 have decreased. This indicates that these participants from the top 3 have at least one tag with which they would not want to work. The next step is also to take into account the workload of people according to the general estimation of tasks and the total number of tasks with the status to do.

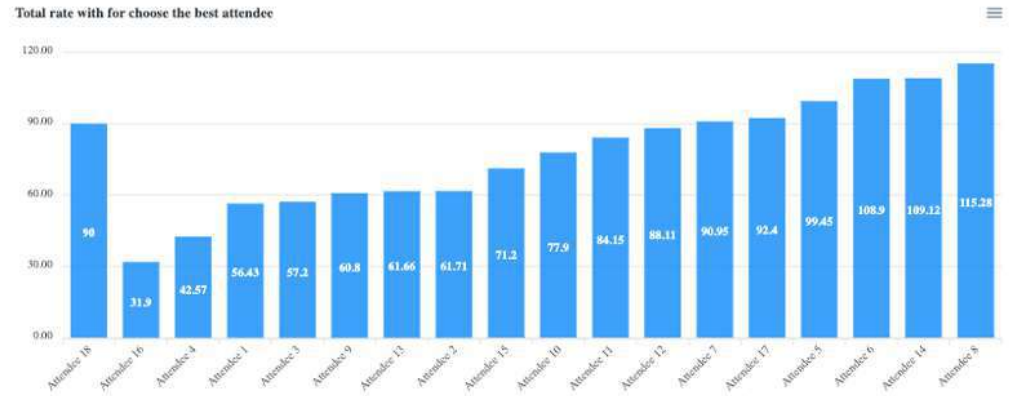


Figure 1.10 – Total rate for choose the best attendee

Taking into account the workload of people according to the general estimation of tasks and total tasks with status to do, we changed the rating and got one general rating as in Figure 8. From it we see that the candidates who were in the previous rating Attendee 5 - 117 points, Attendee 14 - 124 points, and Attendee 8 - 131 points again changed their position Attendee 5 to Attendee 6. Now the top 3 are occupied by participants Attendee 5 - 108.9 points, Attendee 14 - 109.12 points, and Attendee 8 - 115.28 points. As we see, the scores of all participants from the top 3 have decreased. This speaks of the fact that these participants from the top 3 have a certain number of tasks that affects the overall rating, but not significantly. And Attendee 5 is already loaded to a significant extent and cannot be given new tasks. Since there are no participants with the same scores, it can be said from Figure 8 that the most optimal option for task execution is Attendee 8. Since he has 115 points and he has a slight gap with participants Attendee 6 (108) and Attendee 14 (109).

Conclusions:

An automated information system has been proposed for determining the best candidate to perform a task, as well as for classifying the task and adding classification tags to it, which ensures accurate selection of the best performer of the task and performs accurate task classification and determines the corresponding tags. Further research is focused on increasing the accuracy and autonomy of the model and improving the level of classification to the level of subgroups of tags.

The prosed system of distribution and evaluation of tasks in the software development process is aimed at improving the accuracy and autonomy of the model, as well as improving the level of classification to the level of subgroups of tags. The system will use machine learning algorithms to identify the best candidate for a task, as well as

to classify the task and add the corresponding tags. The system will also be able to identify the most suitable tasks for a particular candidate, based on their skills and experience. Additionally, the system will be able to provide feedback to the candidate on their performance, allowing them to improve their skills and become more efficient in their work. The system will also be able to track the progress of the task and provide timely updates to the stakeholders. Finally, the system will be able to provide detailed reports on the performance of the task and the candidate, allowing for better decision-making.

The proposed system of distribution and evaluation of tasks in the software development process for Scrum and Agile methods is designed to provide an automated and accurate way of assigning tasks to the best candidate, as well as classifying the task and adding the corresponding tags. The system is designed to be highly accurate and autonomous, and to provide a level of classification that is accurate down to the level of subgroups of tags. The system is also designed to be highly efficient and to provide a streamlined process for assigning tasks to the best candidate. The system is also designed to be highly flexible and to be able to adapt to changing requirements and tasks. Further research is focused on improving the accuracy and autonomy of the system, as well as improving the level of classification to the level of subgroups of tags. Additionally, research is being conducted to improve the system's ability to handle complex tasks and to provide a more efficient and streamlined process for assigning tasks to the best candidate.

References

1. Sudarkina S., Klimentova M., Anichkina I. Methods and Tools of Strategic Marketing Planning. / Sudarkina S., Klimentova M., Anichkina I. // Bulletin of the National Technical University "Kharkiv Polytechnic Institute" (Economic Sciences), (24), 2019. - Kharkiv: KhNTU, 2019. - Pp. 66-78.
2. Barkova K. Methodological Foundations of Strategic Planning of the Enterprise / Barkova K. // Recommended for printing at the meeting of the Scientific Council of the Kharkiv National Economic University named after S. Kuznetsov, Protocol No. 8 of May 3, 2019. - Kharkiv: KhNEU, 2019. - Pp. 41-49.
3. Kashtalyn I.V. Using Artificial Intelligence Systems to Improve Task Planning Tools / I.V. Kashtalyn, R.V. Kulinyak // III Scientific and Practical Conference of Young Scientists and Students "Intelligent Computer Systems and Networks". November 26, 2020, -Ternopil, Ukraine. Ternopil: ZUNU, 2020. - Pp.13-21.
4. Il'kov A.V. Features of Strategic Planning in Small Enterprises in Conditions of Uncertainty / Il'kov A.V. // Ensuring Sustainable Development of the Agricultural Sector of the Economy: Problems, Priorities, Prospects: Materials of the XI International Scientific and Practical Internet Conference October 29-30, 2020: In 2 vol. - Volume 1. - Dnipro: Publishing and Polygraphic Center "Garant SV", 2020. - Pp.72-74.
5. Ostapenko R.M. Organizational Mechanism of Strategic Planning of Agricultural Enterprises / Ostapenko R.M. // Materials of the II International Scientific and Practical Conference (online form) "Formation and Prospects of Development of Entrepreneurial Structures within the Integration into the European Space" - Poltava, 2019. - 715 p.
6. Kazimir V.V. Information Bases for Building Telecommunication Networks / V.V. Kazimir, V. Litvinov, S.M. Shkarlet, S.V. Zaitsev // Bulletin of Chernihiv State Technical University. - Chernihiv: ChDTU, 2013. - 340 p.
7. Steklov V.K. Information System: A Textbook for Students of Higher Educational Institutions in the Direction of "Telecommunications" / V.K. Steklov, L. Berkman. - K : Technique 2014. - 792 p.
8. Stasev Yu.V. Computer Networks. Technologies and Protocols for Modeling: Tutorial / I.V. Ruban, S.V. Dudenko, O.I. Timochko. - Kh.: KhU P S, 2014. - 359 p.
9. Krivutsa V.G. Management of Telecommunications with the Application of Modern Technologies / V. Krivutsa, V.K.Steklov, L.N.Berckman, B.Kostik, B.Oliynyk, S.Sklyarenko // Textbook for Higher Education Institutions. - K.: Tekhnika, 2007. - 384 p.
10. Steklov V.K. Information System: Textbook for Students of Higher Educational Institutions in the Direction of "Telecommunications" / V.K. Steklov, L. Berckman. - K.: Tekhnika, 2014. - 792 p.
11. Krivutsa V.G. Management of Telecommunications with the Application of Modern Technologies / V.G. Krivutsa, V. Steklov, L. Berckman, B.Ya. Kostik, B. Oliynyk, S.M. Sklyarenko // Textbook for Higher Education Institutions. - K.: Tekhnika, 2007. - 384 p.
12. Gorbatiy, I.V. Telecommunications Systems and Networks. Principles of Operation, Technologies and Protocols: Tutorial / I.V. Gorbatiy, A.V. Bondarev // - Lviv: Publishing House of Lviv Polytechnic, 2016. - 336 p.
13. Sovetov, B.Ya. Modeling Systems: Textbook for Bachelors / B.Ya. Sovetov, S.A. Yakovlev. - 7th ed. - M.: Yurait Publishing, 2015. - 343 p.Chen S. Speaker, environment and channel change detection and clustering via the
14. Klimash, M.M. Modern Transformations in Distributed System Architectures: Monograph / M.M. Klimash, A. Luntovsky, V. Romanchuk. - Lviv-Drohobych: Kolo, 2015. - 328 p.
15. Luntovsky A.O. Stages of Development of Modern Info-Telecommunication Services and Energy Efficiency of Network Technologies / A.O. Luntovsky, P. Gus'kov, A. Masiuk // Bulletin of the National University "Lviv Polytechnic". Series: Radioelectronics and Telecommunications. - Lviv: Publishing House of Lviv Polytechnic, 2014. - No. 796. - P. 131-139.
16. Brooks F. Mythical Man-Month, or How Software Systems Are Created. - St. Petersburg: Symbol-Plus, 2006. - 304 p.

17. Vendrov A.M. Design of Software for Economic Information Systems: Textbook. - 2nd ed., rev. and add. - Moscow: Finances and Statistics, 2005. - 544 p.: ill.
18. Iles P. What is Software Architecture? - Resource: http://www.interface.ru/home.asp?artId=2116
19. Kotzubay I.Yu., Chunaev A.V., Shikov A.N. Basics of Designing Information Systems: Tutorial. - St. Petersburg: ITMO University, 2015. - 206 p.
20. Minukhin S.V., Besedovsky O.M., Znakhur S.V. Methods and Models of Design Based on Modern CASE Tools. Tutorial. - Kharkiv: Vyd. KhNEU, 2008. - 272 p. (Ukrainian).
21. Pushchin M.N. Design of Information Systems: Tutorial. - Moscow: MIET Publishing House, 2008. - 234 p.
22. Design of Information Systems: Textbook / V.I. Grekul, G.N. Denischenko, N.L. Korovkina. - M. Internet-Un-t Information Technologies, 2005. - 304 p.
23. Reengineering of Business Processes. - Resource: https://library.if.ua/book/28/1899.html
24. Sommerville I. Software Engineering, 6th Edition.: Trans. from English. - M.: Williams, 2002. - 624 p.
25. Design Technologies. - Resource: https://studfile.net/preview/3997729/page:5
26. Fowler M. et al. Architecture of Corporate Software Systems. - M.: Williams, 2005. - 576 p.

Okrushko Dmytro – student, Khmelnytski National University, Khmelnytski, Ukraine.
e-mail: okrdima@gmail.com

Kashtalian Antonina Serhiivna – Associate professor of the department, Khmelnytski National University, Khmelnytski, Ukraine.
e-mail: kashtaliana@khnu.edu.ua,
https://orcid.org/0000-0002-4925-9713, Scopus author ID: 57218242499, ResearcherID: AAC-7949-2022, https://scholar.google.com/citations?user=H5LrlPoAAAAJ

ДОВІДКА

ПРО ПРИЙНЯТТЯ СТАТТІ ДО ПУБЛІКАЦІЇ

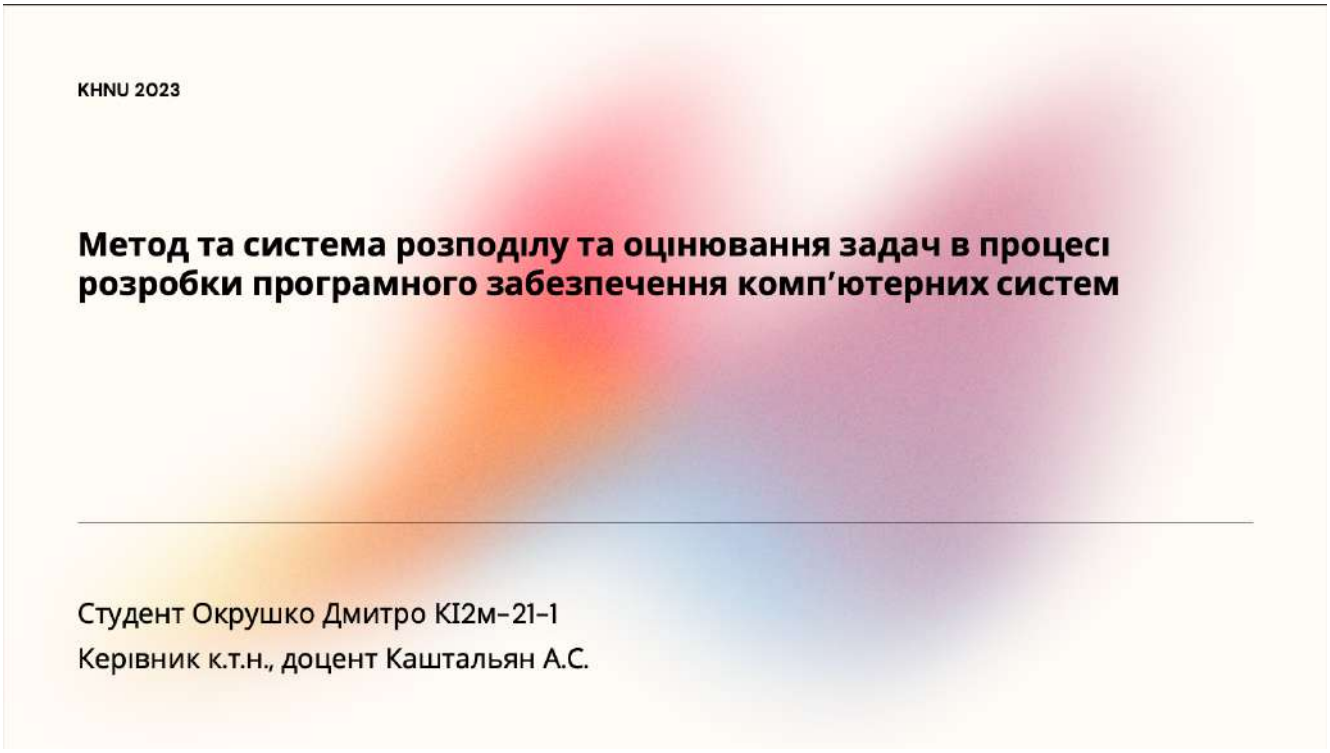
Стаття "System of distribution and evaluation of tasks in the software development process" авторів Д.В. Окрушко, А.С. Каштальян прийнята до публікації в номер 2 фахового наукового журналу категорії Б "Computer systems and information technologies"

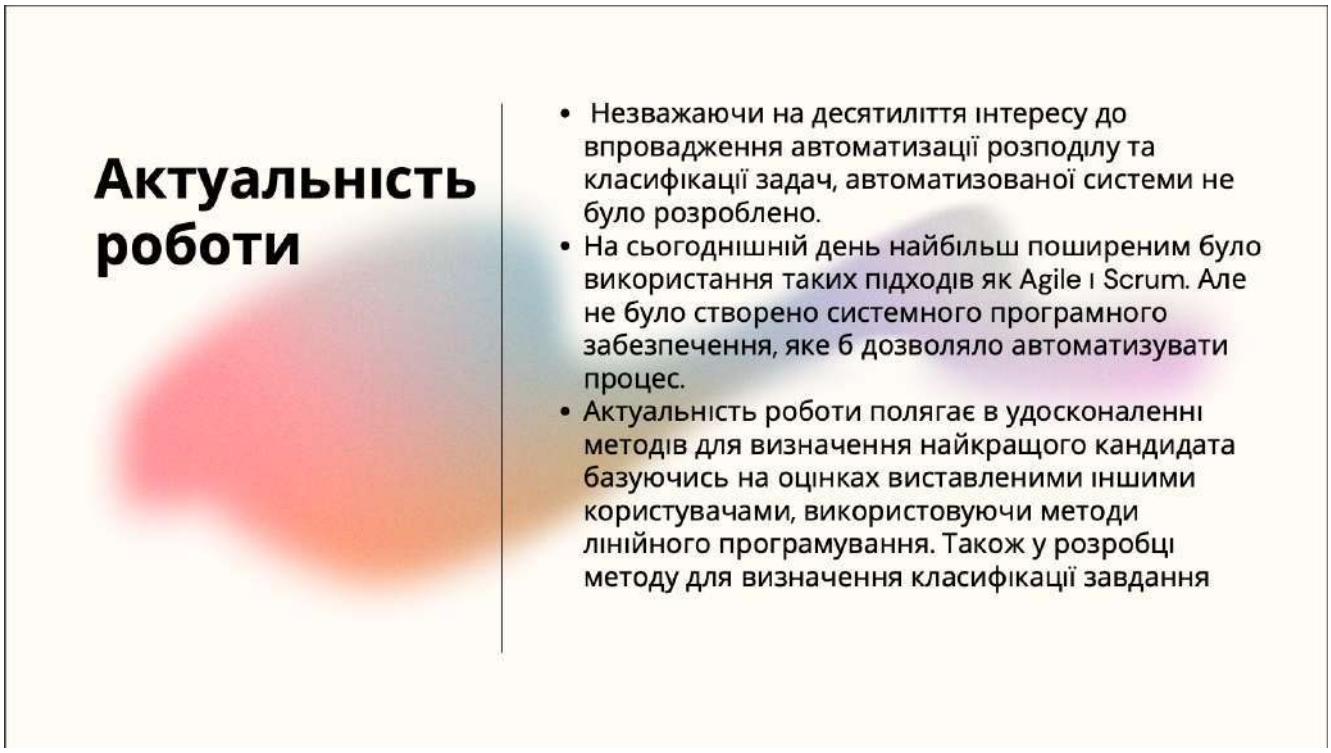
Головний редактор

д.т.н., проф. Говорущенко Т.О.

ДОДАТОК В
(обов'язковий)

ПРЕЗЕНТАЦІЯ ДО ЗАХИСТУ КВАЛІФІКАЦІЙНОЇ РОБОТИ





Дослідження

- **Мета дослідження** – автоматизація та підвищення ефективності процесу розподілення та класифікації задач.
- **Об'єкт дослідження** – процес автоматичного розподілення та класифікації задач під час розробки програмного забезпечення.
- **Предмет дослідження** – метод та інформаційна система для визначення найкращого кандидата на виконання завдання

Наукова новизна результатів

Методи:

- **1.** Удосконалено метод для визначення найкращого кандидата на виконання завдання, який забезпечує, створення системи рейтингів базуючись на оцінках як виставили інші користувач, враховує завантаженість людини, також враховує класифікацію задач на наявність тегів preffered чи not preffered.
- **2.** Розроблено метод для визначення класифікації завдання, який забезпечує пошук групи тегів як належать до конкретного завдання структури завдання з новими класифікаційними тегами.

Практична значимість отриманих результатів

- Значимість полягає у розробці інформаційної системи для визначення найкращого кандидата на виконання завдання та виконувати класифікацію завдань, за допомогою визначених тегів, з використанням стеку MERN і Firebase у програмної платформи для проектованої системи.

Постановка задачі

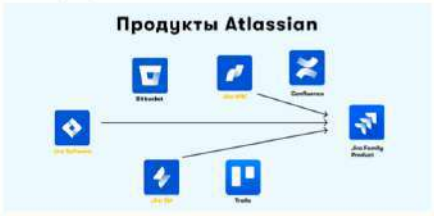
Для розробки системи розподілу та оцінювання задач в процесі розробки системного програмного забезпечення комп'ютерних систем необхідно виконати наступні задачі:

- Дослідити поточні процеси розподілення та класифікації задач в команді.
- Розробка правил для прийняття рішень щодо розпоідлу та класифікації задач.
- Розробка методу для автоматичного розподілення задач між учасниками.
- Розробка методу для автоматичної коасифікації задач на основі класифікаційних тегів.
- Проектування архітектури системного програмного забезпечення комп'ютерних систем для розподілу та оцінюванню задач

Аналіз існуючих рішень

Було проаналізовано такі сервіси:

- Jira
- Asana
- Notion

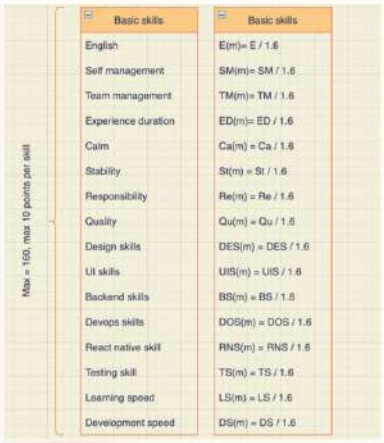


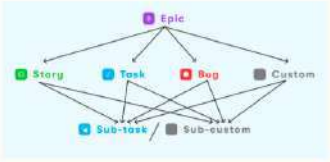
Генерація датасету на базі оцінок користувачів

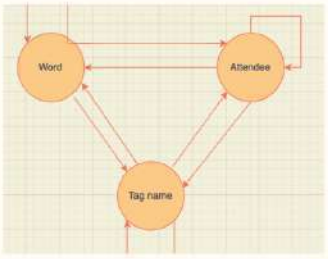
Для того щоб приступити до етапу розробки алгоритму який буде вирішувати задачу оцінювання та класифікацію задачі, необхідно створити відповідний дата сет на основі якого буде будуватися алгоритм. Для цього потрібно визначитись із навичками.До нього буде входити:

1. Рівень англійської.
2. Вміння використовувати час для самого себе.
3. Вміння керувати командою.
4. Досвід у галузі.
5. Спокій користувача.
6. Стабільність.
7. Відповідальність.
8. Якість..
9. Дизайн навички.
10. Інтерфейс користувача.
11. Навички сервера.
12. Навички розгортання.
13. Навички із мобільної платформи.
14. Навички тестування.
15. Швидкість опанування нового матеріалу.
16. Швидкість написання коду.

Генерація датасету на базі оцінок користувачів (2)







Алгоритм розподілення задач

Для того щоб реалізувати алгоритм який буде займатися розподіленням задач між користувачами потрібно на першому етапі створити систему рейтингів. До системи рейтингів буде включено такі параметри:

1. Кількість задач виконано користувачем по всіх тегах які належать до групи тегів.
2. Кількість виконаних задач по всіх підгрупах тегів яка були визначені відповідно до опису задачі.
3. Загальна кількість задач які мають статус To do для кожного із учасників у рейтингу.
4. Загальна кількість задач які мають статус Next для кожного із учасників у рейтингу.
5. Загальна сума всіх оцінок часу на виконання кожного завдання які мають статус To do і Next.
6. Загальна сума оцінок які вистачити всі користувачі один одному для кожного окремого навичку.
7. Рейтинг по S(r).
8. Рейтинг по M(r).
9. Рейтинг по R(r).
10. Рейтинг по D(r).
11. Також сума по всіх решти тегах та навичках які є присутні у задачі після того як було виконано авто класифікацію задачі.

Алгоритм розподілення задач(2)

Для розрахунку впливу preferred tags використаємо формулу :

$$preferred = \prod_i 10 * \{P_i\}/100$$

де P(i) – це скільки разів зустрічається тег у завданні із тегами які є у користувача у полі preferred tags.

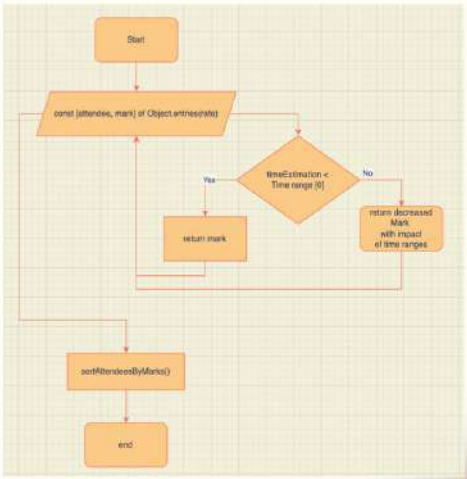
Для розрахунку впливу not preferred tags використаємо такуж формулу.

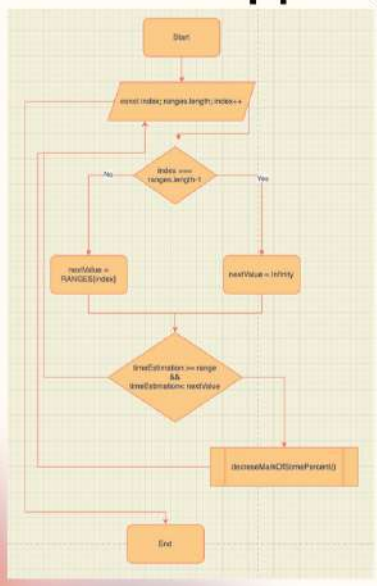
Для того щоб розрахувати саме вплив коефіцієнта саме на оцінку із її зменшенням або збільшенням використаємо формулу:

$$mark = total\ mark + (preferred - not\ preferred)$$

де total mark – це оцінка користувача у результуючому рейтингу.

Алгоритм розподілення задач(3)





Алгоритм класифікації задач

Наступним кроком введемо такі значення як Mandatory tags.
Тобто це ті групи тегів які обов'язково повинні будуть визначені для описаного завдання. Обов'язкових тег груп буде п'ять. Кожна тег група яка зарання визначена або ж створення в процесі використання інтегрованої системи може містити в собі підмножину величиною N.

П'ять основних груп будуть:

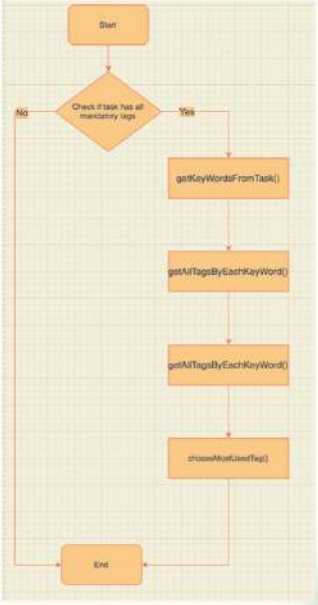
1. Пріоритети. (Priorities).
2. Проекти. (Projects).
3. Необхідні навички. (Skills).
4. Тип. (Types).
5. Фаза виконання задачі. (Phase).

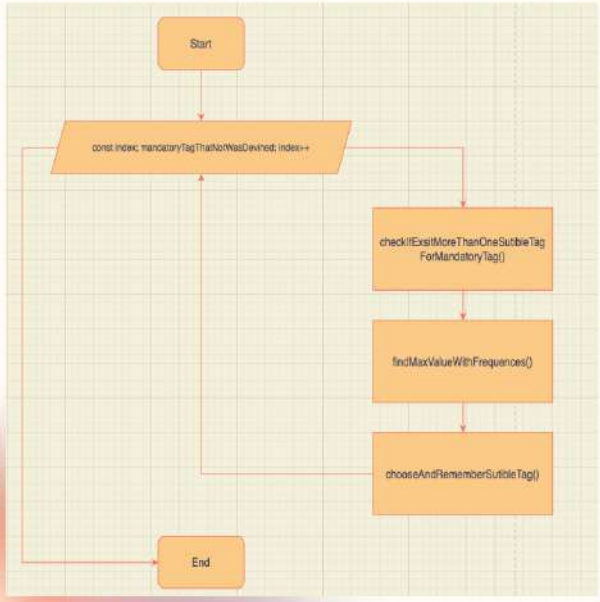
Для визначення тегів які відносяться до задачі будемо в основі використовувати формулу:

$$Tags\{N\} = \sum_{i=1}^{N}(\overrightarrow{max}(Priotities\{SubPriorities(i)\}) + \sum_{i=1}^{N}(\overrightarrow{max}(Projects\{SubProjects(i)\}) + \sum_{i=1}^{N}(\overrightarrow{max}(Phases\{SubPhases(i)\}) + \sum_{i=1}^{N}(\overrightarrow{max}(Skills\{SubSkills(i)\}) + \sum_{i=1}^{N}(\overrightarrow{max}(Types\{SubTypes(i)\})$$

де N – це натуральне число, а SubPriorities Priorities,
Sub Projects Projects, SubPhases Phases, SubSkills Skills

Алгоритм класифікації задач(2)



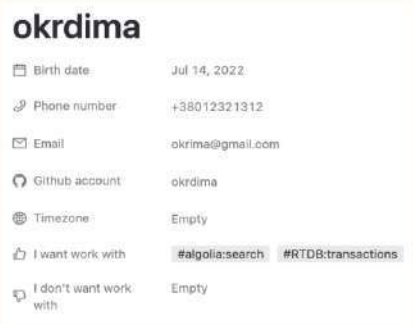


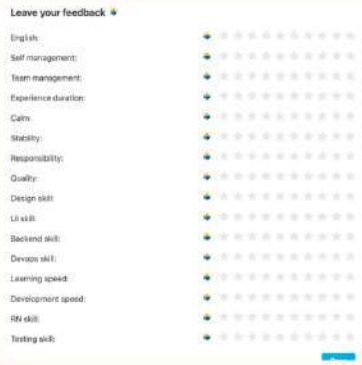
Проектування

Для реалізації поставлених задач було вирішено обрати MERN стек технологій. Із базою даних Firebase.

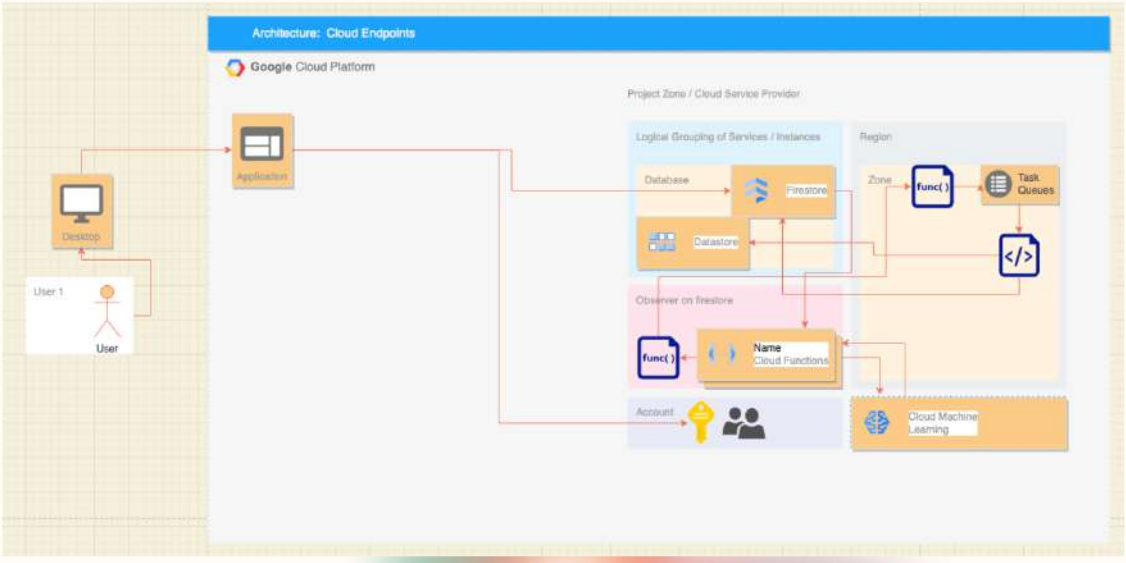
- Запропонована архітектура системи розподілення та класифікації задач спрямована саме на виконання швидкого розподілення задачі на базі тегів які входять до задачі та виконання класифікації задач на базі опису задачі.
- На основі такого розподілу та класифікації задачі система пропнує розподілення задачі із високим процентом точності із врахування завантаженості людини а також на базі preffered and not preferred тегів. А також класифікацію задач із всіма mandatory tags.
- Також система надає навчальний датасет за допомогою якого можна відслідковувати прогрес учасників та визначати похибку системи.

Проектування(2)



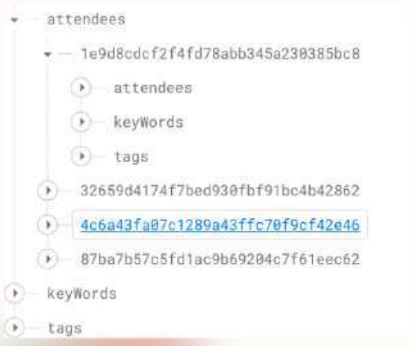


Проектування(3)



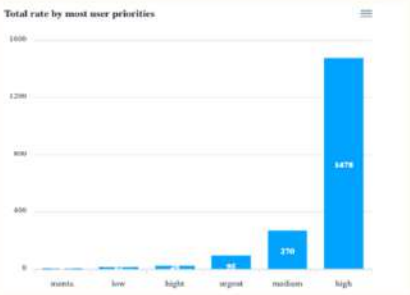
Моделювання

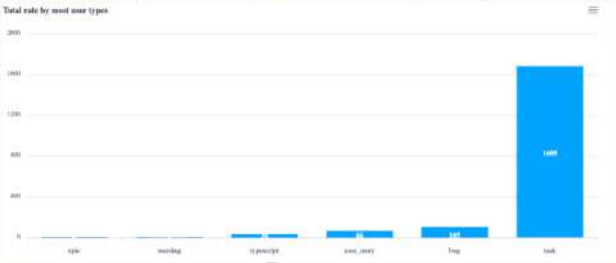
Для візуалізації прогнозу оптимального кандидата візьмемо для прикладу опис завдання – «Я, як супер адміністратор, повинен мати можливість створити форму для запрошення користувачів», і ніяких тегів кваліфікації додаватися не буде.
На першому етапі ми отримаємо схему датасету.

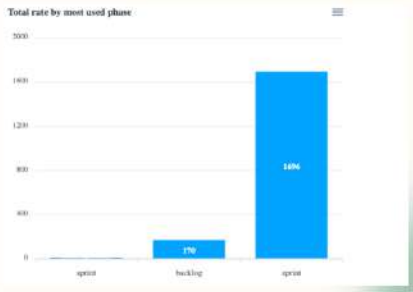


Моделювання(2)

Для опису завдання ми отримали такий набір тегів для кожної групи

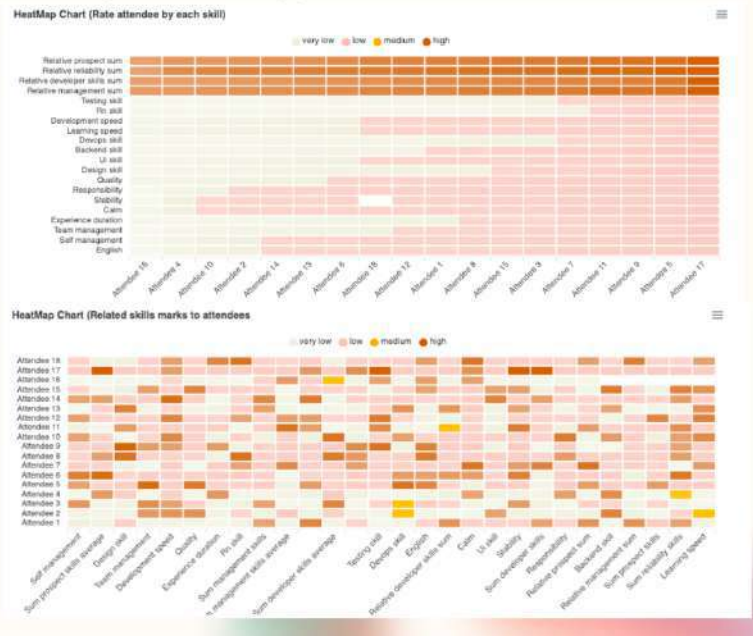






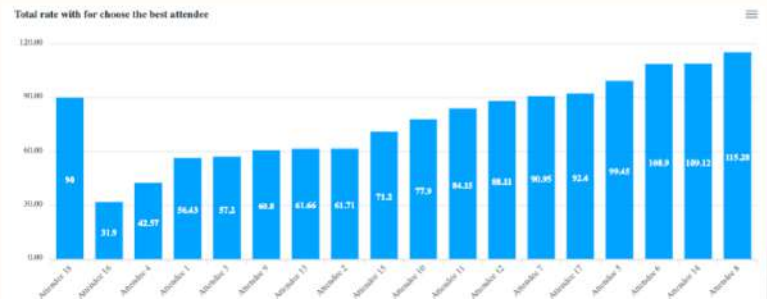
Моделювання(3)

Для опису завдання ми отримали такий набір рейтингів



Моделювання(4)

Для опису завдання ми отримали такий набір рейтингів



Висновки

У роботі за результатми виконаних теоретичних та практичних досліджень забезпечено оптимальний розподіл та класифікації задач на основі системи рейтингів та класифікаційних тегів.

- **У розділі 1 було розглянуто аспекти розробки системи розподілу та оцінки завдань у процесі розробки програмного забезпечення, були досліджені алгоритми які забезпечують максимально точне визначення особи, яка повинна виконати завдання, та відповідні теги класифікації завдань на основі його опису. Були проаналізовані проблем, пов'язані із розподілом завдань і оцінкою в розробці програмного забезпечення. До них належать необхідність точної оцінки завдання, складність забезпечення контролю якості та потреба в ефективній комунікації між розробниками**

Висновки (2)

- **У розділі 2 було розглянуто сучасний стан розподілу та оцінювання завдань. Він розглядався як різноманітні інструменти та методи, доступні для розподілу й оцінки завдань, у тому числі системи відстеження завдань, системне програмне забезпечення для керування проектами та засоби автоматичного тестування. Також було досліджено різні методи, що використовуються для оцінювання завдань.**
- **У розділі 3 було розроблно метод який визначає із опису задачі важливі дієслова та іменники. Метод який генерує дата сет. А також метод який виправляє дата сет у випадку коли передбачення системи було не зовсім точним. Метод який шукає найсильніші відношення у дата сеті між ключовими словами у описі задачі і тегами. Також враховуючи при цьому тег групу базуючись на дереві рішень. Метод який шукає найкращого кандидата для виконання поставленої задачі базуючись на тегах які були відібрані етапом раніше використовуючи лінійне програмування.**

Висновки (3)

- **У розділі 4 розроблено архітектуру системи розподілення та класифікації задач на основі класифікаційних тегів, яка на відміну від існуючих аналогів забезпечує автоматичне розподілення задач серед учасників, та проставляє обов'язкові класифікаційні теги. На підставі рейтингів серед учасників відбувається розподілення задач із високим показником точності.**

Публікації

Окрушко Д.В. Каштальян А.С. СИСТЕМА РОЗПОДІЛЕННЯ ТА ОЦІНЮВАННЯ ЗАДАЧ В ПРОЦЕСІ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

- XIV Всеукраїнська науково-практична конференція "Актуальні проблеми комп'ютерних наук (АПКН – 2022)", м.Хмельницький, ХНУ.
- Кафедральний журнал Computer systems and information technologies №2.

UNICHECK

Ім'я користувача:
Кафедра КІ

ID перевірки:
1014855444

Дата перевірки:
29.04.2023 08:30:56 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
29.04.2023 08:31:15 EEST

ID користувача:
100005591

Назва документа: **Окрушко_Метод та система розподілу та оцінювання задач в процесі розробки програмно...**

Кількість сторінок: **112** Кількість слів: **20460** Кількість символів: **150518** Розмір файлу: **4.16 MB** ID файлу: **1014555369**

2.56%
Схожість

Найбільша схожість: **0.84%** з Інтернет-джерелом (https://www.buh24.com.ua/kanban-vs-scrum-u-chomu-polyagayut-vid...

2% Джерела з Інтернету	67	Сторінка 114
0.76% Джерела з Бібліотеки	55	Сторінка 114

0.19% Цитат

Цитати	3	Сторінка 115
Посилання	1	Сторінка 115

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Замінені символи	4

Sat Apr 29 07:37:40 EEST 2023, Медзатий Дмитро Миколайович, Хмельницький національний університет, ХНУ

Anti-Plagiarism v-15.257

Максимальне співпадіння з одним документом 12.0%

Словники перевірки: en_US, ru_RU, ua_UA. **Помилок в документах: 5%**

	Документ		Сумарний збіг по Базі Даних	
ID: 112731 Назва: Метод та система розподілу та оцінювання задач в процесі розробки програмного забезпечення комп'ютерних систем Додано в БД: 2023-04-29 Автора: Окрушко Д.В. Керівники: Каштальян А.С. Консультанти: Опоненти:	Символи	Лексеми	Символи	Лексеми
	128281	1197	16559 (13%)	160 (13%)

Джерело плагіату

ID	Опис	Наявність плагіату в документі	
		Символи	Лексеми
112065	Назва: ЗВІТ з науково-дослідної практики "Методи розподілення та оцінювання задач в процесі розробки програмного забезпечення комп'ютерних систем" Додано в БД: 2023-03-20 Автора: Д. В. Окрушко Керівники: Гнатчук Є.Г. Консультанти: Опоненти:	15193 (12.0%)	143 (12.0%)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

Дипломник: Окрушко Дмитро Віталійович

Тема: Метод та система розподілу та оцінювання задач в процесі розробки програмного забезпечення комп'ютерних систем

Спеціальність: 123 «Комп'ютерна інженерія»

Обсяг кваліфікаційної роботи:

Кількість сторінок записки 137с.

1. Короткий зміст роботи та прийнятих рішень: Метою кваліфікаційної роботи є підвищення ефективності та автоматизація процесу оцінювання, класифікації та розподілення задач в процесі розробки системного програмного забезпечення.

2. Висновок про відповідність роботи дипломному завданню: Робота повністю відповідає поставленому завданню

3. Характеристика виконання кожного розділу, ступінь використання останніх досягнень науки і техніки і передових методів роботи: В першому розділі проведено аналіз відомих методів оцінювання та класифікації задач розробки програмного забезпечення комп'ютерних систем, визначено їх переваги та недоліки, визначено вимоги до сучасної системи розподілу та класифікації задач, яка інтегрується в систему прийняття рішень. В другому розділі визначені методи та алгоритми, які можуть застосовуватися до класифікації та оцінювання задач, зокрема дерева прийняття рішень, випадковий ліс, лінійне програмування, генетичний алгоритм. В третьому розділі розроблено вдосконалений метод для визначення найкращого кандидата на виконання задачі, який забезпечує створення системи рейтингів на основі оцінок інших користувачів, метод для класифікації задач, ґрунтуючись на повному описі поставленої задачі. Базове представлення дерев прийняття рішень удосконалене системою графів, які мають відношення багато до багатьох, що дозволяє отримати історію відношень та частоти пар слів і тегів навичок та проектів. Також цей метод дозволяє виконувати класифікацію задач за відсутньої історії пар слів та тегів. Вперше розроблено метод для визначення класифікації завдання, який

забезпечує пошук групи тегів, належних до конкретного завдання, розподілення задач на основі лінійного програмування, де цільовою функцією була створена система рейтингів, що враховує завантаженість виконавця та клас задач. Для пошуку найоптимальнішого кандидати додано виплив суми загальних оцінок задач із статусом to do та next та вплив тегів. В четвертому розділі розроблено та реалізовано систему для визначення найкращого кандидата та класифікації задачі та додавання до неї тегів класифікації, що забезпечує оптимальний вибір найкращого виконавця. Розроблена система оцінювання, класифікації та розподілення задач в процесі розробки системного програмного забезпечення дозволяє інтеграцію із Scrum та Agile. Система визначає найкращого кандидата для заданої задачі, визначає набір тегів, є точною та автономною, спрощує процес призначення задач найбільш оптимальному кандидату, є гнучкою та адаптивною до зміни вимог, завдяки чому досягається підвищення ефективності розробки системного програмного забезпечення.

4. Позитивні сторони роботи: отримання двох пунктів наукової новизни

5. Негативні сторони роботи:

6. Оцінка графічного оформлення та пояснювальної записки роботи: Пояснювальна записка оформлена коректно, згідно діючих стандартів оформлення документації

7. Відгук про роботу в цілому: Робота виконана на високому науково-технічному рівні

8. Інші зауваження:

9. Оцінка дипломної роботи: відмінно

Рецензент (прізвище, ім'я, по батькові, посада, місце роботи) _____

Горошко А.В., д.т.н., професор, професор кафедри фізики і електротехніки

_____

"_2_" ____травня_____2023 р. _____ (підпис)

Завідувачу кафедри КІС
д-р.техн.наук, проф. Говорущенко Т. О.

Окрушка Дмитра Віталійовича
ПІБ здобувача вищої освіти

ФІТ, 2 курсу, групи КІ2м-21-1

ЗАЯВА

З правилами чинного Положення «Про систему забезпечення академічної доброчесності у Хмельницькому національному університеті» від 01.07.2022, згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування заходів дисциплінарної та академічної відповідальності, ознайомлений (а). Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на плагіат оповіщений(а) та надаю свою згоду на обробку та збереження університетом моєї роботи в інституційному репозитарії університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-технічних засобів (Unicheck та Anti-Plagiarism) та використання роботи для виявлення плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення плагіату в текстах робіт.

Робота для перевірки університетом надається в друкованому та електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

27 квітня 2023 року

РІШЕННЯ ЕКСПЕРНОЇ КОМІСІЇ

КАФЕДРИ КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ ТА ІНФОМАЦІЙНИХ СИСТЕМ

ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатом звіту подібності щодо роботи, генерованого системою виявлення текстових збігів/ідентичності/схожості:

Назва: Метод та система розподілу та оцінювання задач в процесі розробки програмного забезпечення комп'ютерних систем

Автор: Окрушко Дмитро Віталійович

Спеціальність: 123 – Компютерна інженерія

Освітня програма: освітньо-наукова

Науковий керівник: Каштальян Антоніна Сергіївна, доцент кафедри

Після аналізу звіту подібності зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі, є законними і не є плагіатом. Робота приймається до захисту.	відповідає
2	Виявлені запозичення не є плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи Робота приймається до захисту, але має бути відкоригована. Відкоригований варіант має бути поданий на кафедру за 2 дні до захисту, разом із заявою щодо самостійності виконання письмової роботи та ідентичності друкованої та електронної версії роботи	
3	Виявлені запозичення не є плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнені. Робота може бути допущена до захисту (наступного року) після того як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	

Підтвердження:

Запозичення, виявлені в роботі, є законними і не є плагіатом, оскільки:

1) запозичення розміщені в розділах аналізу існуючих методів, які не описують безпосередньо авторське дослідження і не стосуються результатів роботи;
2) окремі виявлені збіги є загальновживаними фразами або виразами.

Сумарний обсяг всіх запозичень, визначений системою виявлення збігів/ ідентичності/схожості, складає 2.56% і адресується до 67 першоджерела, що, з урахуванням наведених обгрунтувань, відповідає характеру наукового дослідження і свідчить на користь кваліфікаційної роботи.

Керівник роботи _____ А.С. Каштальян

Гарант ОП _____ О. С. Савенко

Завідувач кафедри КІІС _____ Т. О. Говорущенко