

Вінницький національний технічний університет
Міністерство освіти і науки України

Кваліфікаційна наукова
праця на правах рукопису

ПРИЙМАК НАТАЛІЯ ВАСИЛІВНА

УДК 004.8:004.415

ДИСЕРТАЦІЯ

**ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ ПОШУКУ АСОЦІАТИВНИХ
ПРАВИЛ ЩОДО ТРИВАЛОСТІ РОЗРОБКИ ПРОГРАМНОГО
ЗАБЕЗПЕЧЕННЯ**

05.13.06 – інформаційні технології
Технічні науки

Подається на здобуття наукового ступеня кандидата технічних наук

Дисертація містить результати власних досліджень. Використання ідей,
результатів і текстів інших авторів мають посилання на відповідне джерело
_____Н. В. Приймак

Науковий керівник Савчук Тамара Олександрівна
к.т.н., професор

Вінниця - 2020

АНОТАЦІЯ

Приймак Н.В. Інформаційна технологія пошуку асоціативних правил щодо тривалості розробки програмного забезпечення. – Кваліфікаційна наукова праця на правах рукопису.

Дисертація на здобуття наукового ступеня кандидата технічних наук за спеціальністю 05.13.06 «Інформаційні технології». – Вінницький національний технічний університет, Вінниця, 2020.

Дисертаційна робота присвячена підвищенню точності визначення тривалості виконання завдання розробником, а також скороченню тривалості пошуку асоціативних правил (АП) щодо тривалості розробки програмного забезпечення (ПЗ), які для цього використовуються. Це досягається за рахунок використання відповідної інформаційної технології (ІТ). Дана ІТ базується на удосконаленому методі визначення частих предметних наборів Frequent Pattern Growth (FP-Growth), серед яких здійснюється пошук таких правил. Об'єктом дослідження є процес пошуку асоціативних правил щодо тривалості розробки програмного забезпечення. Загальний науковий результат роботи – вирішення актуальної науково-практичної задачі підвищення точності визначення тривалості виконання завдання розробником, а також скорочення тривалості пошуку асоціативних правил (АП) щодо тривалості розробки програмного забезпечення (ПЗ), які для цього використовуються. Такий результат досягається за допомогою використання запропонованої інформаційної технології пошуку таких асоціативних правил.

Оскільки програмне забезпечення відіграє важливу роль в сучасному житті людини, тому й процес його розробки та засоби, необхідні для цього, постійно удосконалюються з метою покращення даного процесу.

В першому розділі дисертаційного дослідження було здійснено аналіз сучасного стану розвитку процесу розробки програмного забезпечення.

Виконано порівняльний аналіз засобів, що використовуються для управління процесом розробки програмного забезпечення, а також розглянуто моделі процесу розробки програмного забезпечення. Зроблено висновок, що відповідні засоби, не дозволяють спрогнозувати тривалість виконання завдання, базуючись на накопичених даних. Також встановлено, що при розробці програмного забезпечення, з використанням розглянутих моделей, враховується лише пріоритет завдання, а тому існує необхідність у створенні інформаційної моделі, що також враховує такі характеристики, як складність, критичність та тривалість виконання завдання.

Було досліджено методи інтелектуального аналізу даних, які використовуються щодо розробки програмного забезпечення, та доведено доцільність їх застосування під час такого процесу. Також проведено дослідження методів визначення частих предметних наборів, які використовуються для пошуку асоціативних правил щодо тривалості розробки програмного забезпечення. Встановлено, що FP-Growth є методом, який доцільно застосовувати для пошуку АП щодо тривалості розробки ПЗ. Даний метод потрібно удосконалити з метою підвищення точності визначення тривалості виконання завдання розробником, а також скорочення тривалості пошуку асоціативних правил, які для цього використовуються; а також його адаптації до відповідної предметної області. Удосконалення полягає у класифікації завдань залежно від їх характеристик (складність, критичність, пріоритет, тривалість виконання завдання) перед пошуком асоціативних правил, що дозволить пришвидшити процес пошуку таких правил.

Здійснено обґрунтування доцільності пошуку і використання асоціативних правил щодо тривалості розробки програмного забезпечення. Підтверджено актуальність наукової та практичної задачі розробки інформаційної технології пошуку асоціативних правил щодо тривалості розробки програмного забезпечення, що базується на удосконаленому

методі визначення частих предметних наборів для пошуку АП FP-Growth.

У другому розділі запропоновано інформаційну модель процесу пошуку асоціативних правил щодо тривалості розробки програмного забезпечення, яку можна подати у вигляді:

$$AR = \langle T, D, minsupp, minconf, \mu, \Omega, NT, PT \rangle,$$

де T – множина завдань, що були виконані під час розробки програмного забезпечення; D – множина розробників, які можуть виконати поставлене завдання; $minsupp$ – мінімальне значення підтримки асоціативних правил; $minconf$ – мінімальне значення достовірності асоціативних правил; μ – метод, за допомогою якого буде здійснюватися пошук асоціативних правил; Ω – множина знайдених асоціативних правил; NT – множина завдань, для яких необхідно визначити тривалість їх реалізації розробником певного рівня кваліфікації; PT – множина завдань з визначеним часом, який необхідний на їх виконання розробником, що маю певну кваліфікацію.

Вхідними даними для інформаційної моделі є множина завдань, серед яких буде здійснюватися пошук асоціативних правил з використанням удосконаленого методу визначення частих предметних наборів FP-Growth; вихідними даними є множина завдань з часом, необхідним розробникам для реалізації таких завдань.

Удосконалено метод визначення частих предметних наборів FP-Growth, що використовуються для пошуку асоціативних правил, за рахунок класифікації вхідної множини завдань на три класи, відповідно до характеристики завдань: завдання категорії 1 (завдання, які легко реалізувати), завдання категорії 3 (завдання, які важко реалізувати), завдання категорії 2 (завдання, по складності між легкими і важкими). Таке удосконалення підвищило точність визначення часу, необхідного на виконання завдання з використанням знайдених асоціативних правил, а також скорочення тривалості пошуку таких залежностей.

Для визначення часу, необхідного на реалізацію завдання розробником певної кваліфікації, розроблено алгоритм використання знайдених асоціативних правил щодо тривалості розробки програмного забезпечення. Це допоможе менеджерам проектів при плануванні та управлінні процесом розробки програмного забезпечення. Наведено UML-діаграму діяльності процесу використання асоціативних правил, знайдених щодо тривалості розробки програмного забезпечення.

У третьому розділі дисертаційного дослідження було розроблено інформаційну технологію пошуку асоціативних правил щодо тривалості розробки програмного забезпечення, що базується на запропонованій інформаційній моделі даного процесу.

Здійснено відображення процесу розробки програмного забезпечення за допомогою марковських ланцюгів з дискретними станами та дискретним часом. Таке відображення дозволило, з використанням єдиного математичного апарату, відобразити даний процес на різних рівнях його деталізації.

Також здійснено моделювання процесу пошуку асоціативних правил щодо тривалості розробки програмного забезпечення з використанням удосконаленого методу визначення частих предметних наборів FP-Growth. Результати моделювання показали зростання швидкості пошуку АП до 2%, якщо кількість завдань для аналізу перевищує 750. Також було встановлено, що точність визначення тривалості виконання завдань залежить від кількості знайдених асоціативних правил: точність зросла у 2,3 рази при збільшенні кількості асоціативних правил у 11 разів. Результати моделювання процесу використання асоціативних правил з метою визначення тривалості виконання завдання розробником певної кваліфікації показали зростання швидкості визначення часу, необхідного на виконання завдання розробником до 5 разів.

У четвертому розділі дисертаційного дослідження розроблено інформаційну систему пошуку асоціативних правил щодо тривалості

розробки програмного забезпечення з використанням запропонованої інформаційної технології. Для цього було наведено рекомендації щодо її розробки, запропоновано її структуру та спроектовано БД відповідного програмного забезпечення.

Аналіз результатів впровадження та використання інформаційної технології пошуку асоціативних правил щодо тривалості розробки програмного забезпечення показав підвищення точності визначення тривалості виконання завдання розробником, а також скорочення тривалості пошуку асоціативних правил, які для цього використовуються. А саме: збільшення точності визначення тривалості виконання завдання в 1,7 рази при збільшенні знайдених асоціативних правил до 100; скорочення тривалості пошуку АП на 1,2% якщо кількість записів, серед яких здійснюється пошук АП, від 500 до 750, та на 2%, якщо кількість записів від 750 до 100; зростання швидкості визначення часу необхідного на виконання завдання розробником певної кваліфікації від 2 до 6 разів в залежності від кількості завдань, для яких цей час визначався.

Ключові слова: програмне забезпечення, процес розробки програмного забезпечення, інформаційна технологія, інформаційна модель, асоціативні правила, метод FP-growth, класифікація, дерева рішень, інтелектуальний аналіз даних.

СПИСОК ПУБЛІКАЦІЙ ЗДОБУВАЧА ЗА ТЕМОЮ ДИСЕРТАЦІЇ

[1] Т. О. Савчук, та Н. В. Приймак, “Обґрунтування доцільності використання методу fp-growth (fpg) при розробці програмного забезпечення”, *Вчені записки ТНУ імені В.І. Вернадського. Серія Технічні науки*, № 5, с. 41 – 45, 2018.

[2] Т. О. Савчук, та Н. В. Приймак, “Розробка інформаційної моделі процесу пошуку асоціативних правил при розробці програмного

забезпечення”, *Інформаційні технології та комп’ютерна інженерія*, vol 42, № 2, с. 43 – 48, 2018.

[3] Т. О. Савчук, та Н. В. Приймак, “Розробка інформаційної технології процесу пошуку асоціативних правил при розробці програмного забезпечення”, *Наукові праці : наук. журн. / Чорном. нац. ун-т ім. П. Могили. Комп’ютерні технології*, вип. 305, т. 317, с. 6 – 11, 2018.

[4] T. O. Savchuk, and N. V. Pryimak, “Modeling of software development process with the Markov processes”, *Eastern-European jornal of Enterprise Technologies . Information technology*, vol. 3, no. 2, pp. 33 – 38, 2017.

[5] T. O. Savchuk, N. V. Pryimak, A. Assembay, T. Zyska, M. Junisbekov, and A. Annabaev “The technology of searching the associative rules while developing the software”, *Proc. SPIE 10445, Photonics Applications in Astronomy, Communications, Industry, and High Energy Physics Experiments*, 2017, doi: 10.1117/12.2280900.

[6] Т. О. Савчук, Н. В. Приймак, “Використання методів інтелектуального аналізу даних при тестуванні програмного забезпечення”, на *X міжнародній конференції «Інтернет-освіта-наука-2016»*, Вінниця, 2016, с. 28 – 29.

[7] Т. О. Савчук, та Н. В. Приймак, “Використання методів штучного інтелекту для прогнозування часу для виправлення дефектів розробки програмного забезпечення”, на *X міжнародній конференції «Інтернет-освіта-наука-2016»*, Вінниця, 2016, с. 30 – 31.

[8] Т. О. Савчук, та Н. В. Приймак, “Обґрунтування вибору методу визначення частих предметних наборів для пошуку асоціативних правил при розробці програмного забезпечення”, на *XI міжнародній конференції «Інтернет-освіта-наука-2018»*, Вінниця, 2018, с. 45 – 46.

[9] Т. О. Савчук, та Н. В. Приймак, “Інформаційна модель процесу пошук і використання асоціативних правил при розробці програмного забезпечення”, на *XI міжнародній конференції «Інтернет-освіта-наука-2018»*, Вінниця, 2018, с. 43 – 44.

[10] T. O. Savchuk, and N. V. Pryimak, “Feasibility study of using associative rules during the software development”, on *IX international Conference of the Georgian Mathematical Union*, Batumi, 2018, pp. 203 – 204.

[11] Т. О. Савчук, та Н. В. Приймак, “Інформаційна технологія пошуку асоціативних правил при розробці програмного забезпечення”, на *Всеукраїнській наук.-практ. конф. здобувачів вищої освіти й молодих учених*, Кропивницький, 2018, с. 100 – 102.

[12] Т. О. Савчук, та Н. В. Приймак, “Використання frg-алгоритму для пошуку асоціативних правил при прийнятті рішень в управлінні процесами”, на *XLV НТК проф.-викл. складу, співробітників та студентів університету*, Вінниця, 2016, с. 929 – 930.

[13] Т. О. Савчук, та Н. В. Приймак, “Використання Марковських ланцюгів при моделюванні процесу розробки програмного забезпечення”, на *XLVI НТК проф.-викл. складу, співробітників та студентів університету*, Вінниця, 2017, с. 1019 – 1020.

[14] Т. О. Савчук, та Н. В. Приймак, “Метод пошуку асоціативних правил при розробці програмного забезпечення, що базується на попередньому розбитті множини завдань” на *XLVIII НТК проф.-викл. складу, співробітників та студентів університету*, Вінниця, 2019, с. 655 – 656.

[15] Т. О. Савчук, Н. В. Приймак, “Інформаційна система пошуку асоціативних правил при розробці програмного забезпечення”, *Свідоцтво про реєстрацію авторського права на твір №82173*; Жов. 11, 2018.

ABSTRACT

Pryimak N. V. The information technology of the associative rules search for the duration of software development – Qualification research paper, manuscript copyright.

Thesis for the degree of a candidate of technical sciences in specialty 05.13.06 «Information technology». – Vinnytsia National Technical University, Vinnytsia, 2020.

The thesis is dedicated to the study of increasing the accuracy of determining the duration of the task development, as well as of reducing the duration of the associative rules search, which are used for this. It can be reached by the development and implementation of relevant information technology. This information technology is based on the advanced FP-Growth method of generating frequent pattern sets, among which, the search of such rules is performed. The object of the research is the process of the associative rules search for the duration of software development. The general scientific result of the work is a solution for the actual scientific and practical task of increasing the accuracy of determining the duration of the task development, as well as of reducing the duration of the associative rules search which are used for this, due to the development and implementation of relevant information technology.

As the software plays a significant role in the modern life, the process of its development and the tools necessary, used for this purpose, are constantly being developed to improve this process.

In **the first section** of the thesis, an analysis of the current state of the software development process has been performed.

A comparative analysis of the tools, used for the software management process was done, and also the models of the software development process was reviewed. It is concluded that the reviewed tools do not allow to predict the duration of the task, based on the accumulated data. It was also found that usage of considered models during the software development, take into account only the

priority of the task, and therefore there is a need to create an information model, which will also take into account such characteristics as the complexity, criticality and the duration of task implementation.

It is established that during software development, usually several models are used, and they have defined stages. Appropriate tools are used to manage this process, but they do not allow to predict the duration of the task implementation, based on the accumulated data.

The methods of the data mining, used during the software development, were investigated, and the feasibility of their use was proved. Also, the methods of generating frequent pattern sets, which are used for associative rules search for the duration of software development, were studied. It was found that FP-Growth is the method that should be used for associative rules search for the duration of software development. This method needs to be improved in order to increase the accuracy of determining the duration of the task development, as well as to reduce the duration of the associative rules search which are used for this, and as well as its adaptation to the relevant subject area. The improvement is based on the data classification before the associative rules search, which will allow to accelerate the process of such rules searching.

The feasibility of searching and using associative rules for the determining the duration of software development is substantiated. The relevance of the scientific and practical task of developing information technology of searching associative rules for the software development duration, that is based on an advanced method of determining frequent pattern sets FP-Growth, is confirmed.

In **the second section** an information model of the associative rules search for the duration of software development is proposed. This model can be submitted as a set:

$$AR = \langle T, D, minsupp, minconf, \mu, \Omega, NT, PT \rangle,$$

where T is the set of tasks performed during the software development; D is a set of developers, who can solve the task; $minsupp$ is the minimum value of

the associative rules support; $minconf$ is the minimum value of the associative rules confidence; μ is a method which will be used to search the associative rules; Ω is the set of the found associative rules; NT is a set of tasks, for which it is necessary to determine the duration of their implementation by a developer with particular level of proficiency; PT is a set of tasks with the defined time necessary for their implementation by a developer with particular professional skill.

The inputs for this information model is a set of tasks, among which the associative rules will be searched by using the advanced FP-Growth method of generating frequent pattern sets; the outcome data is a set of tasks with the defined time necessary for developers to solve them.

The method FP-Growth of the generation of the frequent pattern sets used for the associative rules search is improved by classifying the input set of tasks into three classes, depending on their characteristics: tasks of category 1 (tasks that are easy to implement), tasks of category 3, which are difficult to implement, tasks of category 2 (tasks, with the complexity between category 1 and category 3). This improvement helped to increase the accuracy of determining the duration of the task development, as well as to reduce the duration of the search of the associative rules, which are used for this.

To determine the time required for the task implementation by a specific developer, an algorithm of using the found associative rules for the duration of software development was created. It will help project managers with planning and managing the software development process. The UML activity diagram, which describes the process of using such associative rules, was introduced.

In **the third section** of the research, an information technology of the associative rules search for the duration of software development was created. It's based on the proposed information model of this process. The software development process was demonstrated through the use of Markov chains with discrete states and discrete time. Such simulation allows to use a single

mathematical apparatus to display this process at various levels of its details.

The simulation of the associative rules search for the duration of software development by using the advanced FP-Growth method for generating frequent pattern sets has been implemented. The simulation results showed an increase of the speed of the associative rules search to 2%, in case the number of tasks for analysis exceed 750. It was also found that the accuracy of predicting the tasks duration depends on the number of found associative rules: accuracy increased in 2.3 times with an increase of associative rules number in 11 times. The results of modeling the process of using associative rules to determine the duration of the task development by the developer of a certain qualification, showed an increase of speed to determine the time, required to complete the task, up to 5 times.

In **the fourth section** of the thesis the information system for the associative rules search for the duration of software development was developed, using the proposed information technology. To archive this, recommendations for its development were given, its structure was proposed, and a database of the related software was designed.

The analysis of the results of the integration and use of information technology of the associative rules search for the duration of software development showed: an increase of the accuracy of the prediction of the task implementation duration in 1,7 times with the increase of found associative rules to 100; a decrease of the search duration of the associative rules by 1.2% if the number of records, among which they are searched, is between 500 and 750, and by 2% if the number of records is from 750 to 1000; increase in the speed of determining the time required to complete the task by the developer of a certain qualification from 2 to 6 times, depending on the number of tasks for which this time was determined.

Keywords: software, the process of software development, information technology, information model, associative rules, method FP-growth, classification, decision trees, data mining.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ.....	15
ВСТУП.....	16
РОЗДІЛ 1 СУЧАСНІ МОДЕЛІ, МЕТОДИ ТА ЗАСОБИ ПІДТРИМКИ ПРОЦЕСІВ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ.....	23
1.1 Аналіз засобів, що використовуються для управління процесом розробки програмного забезпечення	23
1.2 Сучасні моделі розробки програмного забезпечення.....	30
1.3 Особливості використання методів інтелектуального аналізу даних щодо розробки програмного забезпечення.....	35
1.4 Обґрунтування доцільності пошуку і використання асоціативних правил щодо тривалості розробки програмного забезпечення.....	40
1.5 Постановка задачі.....	54
1.6 Висновки до розділу 1.....	55
РОЗДІЛ 2 РОЗРОБКА ТЕОРЕТИКО-МЕТОДИЧНИХ ЗАСАД ПРОЦЕСУ ПОШУКУ АСОЦІАТИВНИХ ПРАВИЛ ЩОДО ТРИВАЛОСТІ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ.....	57
2.1 Інформаційна модель процесу пошуку асоціативних правил щодо тривалості розробки програмного забезпечення	57
2.2 Удосконалений метод FP-Growth визначення частих предметних наборів для пошуку асоціативних правил	66
2.3 Алгоритм використання асоціативних правил для визначення часу, необхідного для виконання завдання розробником.....	73
2.4 Висновки до розділу 2.....	78
РОЗДІЛ 3 ПОШУК АСОЦІАТИВНИХ ПРАВИЛ ЩОДО ТРИВАЛОСТІ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ З ВИКОРИСТАННЯМ ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ.....	80
3.1 Інформаційна технологія пошуку асоціативних правил щодо тривалості розробки програмного забезпечення	80
3.2 Моделювання процесу розробки програмного забезпечення.....	84
3.3 Моделювання процесу пошуку асоціативних правил щодо тривалості розробки програмного забезпечення з використанням удосконаленого методу FP-Growth	93

	14
3.4 Висновки до розділу 3.....	101
РОЗДІЛ 4 ІНФОРМАЦІЙНА СИСТЕМА ПОШУКУ АСОЦІАТИВНИХ ПРАВИЛ ЩОДО ТРИВАЛОСТІ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ	103
4.1 Рекомендації щодо розробки інформаційної системи пошуку асоціативних правил щодо тривалості розробки програмного забезпечення.....	103
4.1.1 Структура інформаційної системи.....	103
4.1.2 База даних програмного забезпечення пошуку асоціативних правил щодо тривалості розробки	112
4.1.3 Графічний інтерфейс користувача інформаційної системи пошуку асоціативних правил.....	115
4.2 Результати впровадження інформаційної технології пошуку асоціативних правил щодо тривалості розробки програмного забезпечення	118
4.3 Висновки до розділу 4.....	123
ВИСНОВКИ.....	125
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	128
ДОДАТКИ.....	143
Додаток А Акти впровадження результатів дисертації.....	144
Додаток Б Список публікацій за темою дисертації.....	146
Додаток В Ймовірності переходів між станами для трьох рівнів деталізації процесу розробки програмного забезпечення.....	148
Додаток Г Лістинг програмної реалізації основних модулів програмного забезпечення пошуку асоціативних правил щодо тривалості розробки програмного забезпечення.....	149
Додаток Д Інструкція користувача інформаційної системи пошуку асоціативних правил щодо тривалості розробки програмного забезпечення.....	156
Додаток Е Вибірка завдань для пошуку асоціативних правил.....	162

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

АП	Асоціативне правило
БД	База даних
ІАД	Інтелектуальний аналіз даних
ІТ	Інформаційна технологія
ІМ	Інформаційна модель
ОО мова	Об'єктно-орієнтовна мова
ПЗ	Програмне забезпечення
FP-Growth	Frequent Pattern Growth
UML	Unified modelling language

ВСТУП

Обґрунтування вибору теми дослідження

У сучасному світі програмне забезпечення (ПЗ) використовується для роботи, навчання, спілкування; для здійснення покупок, платежів, подорожей тощо. Програмне забезпечення має важливе значення для різних галузей бізнесу та виробництва: використовується в банках та страхових компаніях; в енергетиці та сільському господарстві; у торгівлі, медицині, навчанні тощо.

Оскільки програмне забезпечення стає все більш важливим у всіх галузях промисловості та послуг, то процес його розробки та інженерні засоби, необхідні для цього, перебувають у постійному розвитку [1]. Так як програмні продукти різноманітні та мають різний контекст використання: застосування прикладного програмного забезпечення в інформаційній системі чи програмного забезпечення, вбудованого в інші продукти, або ігрового програмного забезпечення тощо, то виникає необхідність оцінки відповідності системи, компонента чи процесу визначеними вимогам: функціональним чи не функціональним [2].

Однією з нефункціональних вимог до застосування програмного забезпечення, згідно зі стандартом IEEE Computer 1985, є вимога до його життєвого циклу: визначення обмежень щодо задіяних людських ресурсів та тривалості процесу розробки такого ПЗ [3], [4]. Точна оцінка графіка закінчення розробки програмного забезпечення необхідна менеджерам для планування роботи команди розробників та ефективного управління даним процесом. Окрім вказаної вимоги важливим показником є організація процесу розробки та якість кінцевого програмного забезпечення.

Одним із способів перевірки, що програмне забезпечення надійне – є контроль усіх етапів його розробки. Традиційна модель розробки програмного забезпечення передбачає перевірку кожного етапу розробки (вимоги до документації, проектування, аналіз, кодування, перевірка та тестування) перед переходом до наступного [5]. Сучасні програмні проекти

розробляються більш динамічно, із застосуванням більш гнучких технологій та методологій розробки; але етапи, перераховані вище, присутні в будь-якому процесі, оскільки усе програмне забезпечення повинно бути розроблене, інтегроване, перевірене, випущене та впроваджене [6].

У зв'язку з тим, що складність і різноманітність програм постійно зростає, то їх якість має важливе значення для споживача. В процесі написання програмного забезпечення його код часто змінюється з метою поліпшення функціональності або для виправлення помилок, які викликають збій у його роботі. Використання систем управління версіями програмного коду та баз даних (БД) завдань, що ставляться під час розробки ПЗ, дозволяють зібрати інформацію про усі зміни програмного коду та використати її для подальшого аналізу [7]. Наявні бази даних містять значну кількість інформації, яку неможливо проаналізувати вручну, а, отже, виникає необхідність у автоматизованому аналізі. Для вирішення даної задачі використовують різноманітні методи інтелектуального аналізу даних (ІАД) до яких належить і пошук асоціативних правил (АП). Такий метод дозволяє знаходити асоціативні залежності, які можна використати для визначення часу, необхідного для реалізації конкретного завдання розробником. Отримана інформація може бути застосована менеджерами проектів для планування розробки ПЗ.

Сучасні методи пошуку асоціативних правил щодо тривалості розробки програмного забезпечення розглянуті в роботах Г. П'ятецький-Шапіро [8], В. Жоу [9], Ф. Жанг [10], Х. Жанг [11], П. Бхатачар'я [12], А. Ламканхфі [13], Е. Гігер [14]. Питання моделювання систем представлено у роботах А. А. Молчанова [15], В.І. Скуріхіна [16], Н. Бусленко [17], В. М. Трояновського [18], Р. Н. Кветного [19], В. М. Дубового [20], Б. І. Мокіна [21], В. Б. Мокіна [22], О. В. Бісікала [23] та інших дослідників. Один із підходів до вирішення задачі прогнозування трудомісткості розробки програмних систем запропоновано у роботі С. Д. Штовби [24].

Отже, наукова задача створення інформаційної технології (ІТ) пошуку

асоціативних правил щодо тривалості розробки програмного забезпечення є актуальною, оскільки використання такої інформаційної технології дозволить підвищити швидкість важливого процесу розробки програмного забезпечення.

Зв'язок роботи з науковими програмами, планами, темами

Дисертаційна робота виконана відповідно до напрямку наукових досліджень кафедри комп'ютерних наук Вінницького національного технічного університету 22 К1 «Моделі, методи, технології та пристрої інтелектуальних інформаційних систем управління, економіки, навчання та комунікацій».

Результати дисертаційного дослідження увійшли у звіт з держбюджетної науково-дослідної роботи на тему «Розробка інформаційної системи пошуку асоціативних правил щодо тривалості розробки програмного забезпечення» № 47/5В (номер договору про творчу співпрацю №47/5 від 15 січня 2018 року), де здобувач був у складі авторів проміжних та заключного звітів. Також впроваджено інформаційну технологію пошуку асоціативних правил щодо тривалості розробки програмного забезпечення в ТОВ «Бізнес-Програми». Відповідні акти впровадження наведено в додатку А.

Мета і завдання дослідження

Метою дисертаційного дослідження є підвищення точності визначення тривалості виконання завдання розробником, а також скорочення тривалості пошуку асоціативних правил, які для цього застосовуються, за рахунок використання відповідної інформаційної технології.

Для досягнення поставленої мети необхідно розв'язати такі *завдання*:

1. Дослідити сучасний стан розвитку процесу розробки програмного забезпечення та проаналізувати існуючі методи визначення частих предметних наборів для пошуку асоціативних правил щодо тривалості даного процесу. Розробити інформаційну модель (ІМ) процесу пошуку асоціативних правил щодо тривалості розробки програмного забезпечення.

2. Удосконалити метод визначення частих предметних наборів FP-

Growth для пошуку асоціативних правил щодо тривалості розробки програмного забезпечення.

3. Розробити інформаційну технологію, яка проводитиме пошук асоціативних правил щодо тривалості розробки програмного забезпечення з метою їхнього подальшого використання при розробці програмного забезпечення.

4. Провести моделювання процесу пошуку асоціативних правил з використанням удосконаленого методу FP-Growth.

5. Розробити інформаційну систему пошуку асоціативних правил щодо тривалості розробки програмного забезпечення, що базується на запропонованій інформаційній технології.

Об'єктом дослідження є процес пошуку асоціативних правил щодо тривалості розробки програмного забезпечення.

Предметом дослідження є інформаційна технологія пошуку асоціативних правил щодо тривалості розробки програмного забезпечення.

Методи дослідження

Використано теоретичні та змішані методи, серед них: методи системного аналізу (порівняльний та структурний аналіз) для визначення властивостей процесу пошуку асоціативних правил щодо тривалості розробки програмного забезпечення; методи інтелектуального аналізу даних (пошук асоціативних правил та класифікацію даних) для розробки інформаційної моделі вищевказаного процесу та удосконаленого методу визначення частих предметних наборів для пошуку асоціативних правил, що базується на цій моделі; методи об'єктного моделювання процесу пошуку асоціативних правил, а також розробки програмного забезпечення; комп'ютерне моделювання для розробки програмного забезпечення пошуку асоціативних правил.

Наукова новизна отриманих результатів

1. Уперше запропоновано інформаційну модель процесу пошуку асоціативних правил щодо тривалості розробки програмного забезпечення,

яка, на відміну від існуючих, враховує такі характеристики завдань (завдань, які потрібно вирішити при розробці програмного забезпечення), як: складність, пріоритет, критичність та тривалість їх виконання. Це дозволило підвищити точність визначення часу, необхідного на виконання завдання розробником певної кваліфікації.

2. Удосконалено метод визначення частих предметних наборів для пошуку асоціативних правил FP-Growth щодо тривалості розробки програмного забезпечення за рахунок введення етапу класифікації завдань відповідно до таких характеристик завдань: складність, пріоритет, критичність та тривалість їх виконання. Це дозволило скоротити час, необхідний для пошуку асоціативних правил щодо тривалості розробки програмного забезпечення, які в подальшому використовуються для визначення часових характеристик реалізації завдання розробником певної кваліфікації.

3. Уперше розроблено інформаційну технологію пошуку асоціативних правил щодо тривалості розробки програмного забезпечення в основі якої, на відміну від існуючих, використовується інформаційна модель та удосконалений метод визначення частих предметних наборів для пошуку асоціативних правил FP-Growth, що дозволило збільшити точність визначення тривалості виконання завдання розробником певної кваліфікації та скоротити тривалість пошуку асоціативних правил щодо тривалості розробки програмного забезпечення.

Практичне значення отриманих результатів

Практична цінність результатів дисертаційного дослідження полягає в тому, що:

- розроблено алгоритм, що реалізує удосконалений метод визначення частих предметних наборів для пошуку асоціативних правил щодо тривалості розробки програмного забезпечення за рахунок введення етапу класифікації завдань відповідно до їх характеристик (складність, пріоритет, критичність та тривалість їх виконання), що дозволило скоротити тривалість пошуку таких

асоціативних правил;

- розроблено алгоритм використання знайдених асоціативних правил щодо тривалості розробки програмного забезпечення, що дозволило автоматизувати процес розробки програмного забезпечення та підвищити точність визначення тривалості виконання завдання розробником певної кваліфікації;

- розроблено інформаційну систему для пошуку асоціативних правил щодо тривалості розробки програмного забезпечення, що реалізує розроблену інформаційну технологію.

Отримані на основі наукових досліджень практичні результати впроваджено в ТОВ «Бізнес-Програми» у вигляді інформаційної системи пошуку асоціативних правил при розробці програмного забезпечення (акт впровадження № 47/5В від 17 вересня 2018 року), а також інформаційної технології пошуку асоціативних правил щодо тривалості розробки програмного забезпечення (акт впровадження від 27 липня 2018 року). Це дозволило приймати ефективні управлінські рішення щодо планування та управління розробкою програмного забезпечення. Впровадження результатів дослідження підтверджено відповідними актами в додатку А.

Особистий внесок здобувача

Основні теоретичні та експериментальні дослідження та висновки дисертаційної роботи були отримані автором особисто.

У друкованих працях, опублікованих у співавторстві, автору дисертації належать: аналіз доцільності використання асоціативних правил щодо тривалості розробки програмного забезпечення [25], [26], аналіз доцільності використання методів ІАД при тестуванні програмного забезпечення [27] – [29], аналіз доцільності використання методу FP-Growth щодо тривалості розробки програмного забезпечення [30], [31], розробка інформаційної моделі процесу пошуку асоціативних правил щодо тривалості розробки програмного забезпечення [32], [33], удосконалення методу пошуку асоціативних правил FP-Growth, за рахунок класифікації даних [34],

представлення процесу розробки програмного забезпечення з використанням марковських процесів [35], [36], розробка інформаційної технології процесу пошуку асоціативних правил щодо тривалості розробки програмного забезпечення [37], [38], комп'ютерне моделювання [39].

Апробація матеріалів дисертації

Результати проведеного дослідження були апробовані на 10-ій та 11-ій міжнародних науково-практичних конференціях «Інтернет – Освіта – Наука» (2016, 2018), XLV, XLVI, XLVIII науково-технічних конференціях професорсько-викладацького складу, співробітників та студентів ВНТУ (2016, 2017, 2019), IX міжнародній науково-практичній конференції Грузинської математичної асоціації, Тбілісі, Грузія (2018), Всеукраїнській науково-практичній конференції здобувачів вищої освіти й молодих учених «Комп'ютерна інженерія і кібербезпека: досягнення та інновації», Кропивницький (2018).

Публікації

Матеріали досліджень опубліковані в 15 наукових працях: 1 стаття, що опублікована в іноземному фаховому журналі, який входить до наукометричної бази SCOPUS, 4 статті, що опубліковані в наукових виданнях з переліку фахових видань України, з них, в тому числі, 1 стаття у журналі, який входить до наукометричної бази SCOPUS; результати дослідження апробовано та опубліковано у вигляді 9 тез на шістьох науково-технічних конференціях (4 публікації у збірниках праць, 1 публікація у збірнику матеріалів конференції, 3 – у вигляді тез доповідей) та одній науково-практичній конференції; отримано 1 свідоцтво на реєстрацію авторського права на комп'ютерну програму (додаток Б).

Структура та обсяг дисертації

Дисертаційна робота складається зі вступу, чотирьох розділів, висновків, списку використаних джерел та додатків. Основний зміст роботи викладено на 108 сторінках друкованого тексту, містить 34 рисунки та 16 таблиць. Список використаних джерел містить 161 найменування. Загальний обсяг роботи становить 164 сторінки.

РОЗДІЛ 1

СУЧАСНІ МОДЕЛІ, МЕТОДИ ТА ЗАСОБИ ПІДТРИМКИ ПРОЦЕСІВ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Під час аналізу сучасного рівня розвитку інформаційних технологій пошуку асоціативних правил щодо тривалості розробки програмного забезпечення можна виділити такі задачі:

- аналіз засобів, що використовуються для управління процесом розробки програмного забезпечення;
- аналіз сучасних моделей розробки програмного забезпечення;
- дослідження доцільності використання методів інтелектуального аналізу даних щодо тривалості розробки програмного забезпечення;
- оцінювання доцільності пошуку і використання асоціативних правил щодо тривалості розробки програмного забезпечення.

1.1 Аналіз засобів, що використовуються для управління процесом розробки програмного забезпечення

Програмне забезпечення являє собою комп'ютерні програми та дані, що зберігаються у цифровому вигляді та використовуються для вирішення визначених задач певного класу [40], [41].

Управління розробкою програмного забезпечення – це планування, реалізація, відслідковування і контроль за проектами з розробки програмного забезпечення [42], [43].

Одним з інструментів управління розробкою ПЗ є мережеве планування, що являє собою метод управління, заснований на застосуванні математичного апарату теорії графів для відображення і алгоритмізації комплексу взаємопов'язаних робіт, заходів або дій для досягнення визначеної мети [44], [45].

Ціллю управління розробкою програмного забезпечення є мінімізація витрат і дотримання термінів розробки. Для її досягнення необхідно вирішити такі задачі:

- отримати максимальну матеріальну вигоду, що може бути досягнуто за рахунок скорочення до мінімуму тривалості процесу розробки ПЗ (за рахунок скорочення загального часу проектування і реалізації);
- максимально ефективно розподілити задіяні трудові ресурси і засоби на час розробки ПЗ.

Моделі мережевого планування, в яких взаємна послідовність і тривалість робіт задані однозначно, називаються детермінованими [46]. До них належать: мережевий графік та діаграма Ганта.

Мережевий графік є графічною формою подання послідовності робіт при розробці програмного забезпечення, яка розкриває внутрішні зв'язки проекту [47], [48]. Приклад мережевого графіку процесу розробки програмного забезпечення наведено на рис. 1.1.

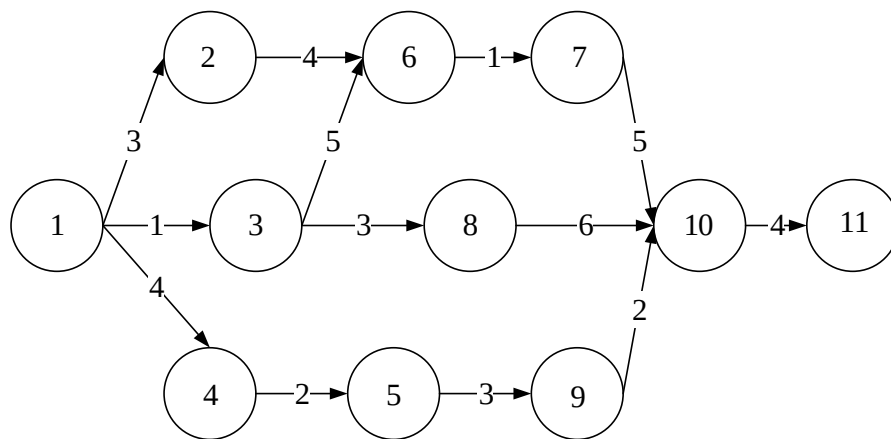


Рисунок 1.1 – Мережевий графік процесу розробки програмного забезпечення

Вершини даного графу – це події, що відображають результат реалізації попереднього завдання, а дуги – це завдання, які потрібно виконати під час розробки програмного забезпечення, та які характеризуються певною тривалістю реалізації. Так відповідно до мережевого графіку наведеного на

рис. 1.1: події 2, 3 і 4 настають, після виконання завдань, тривалістю 3, 1 та 4 умовні одиниці відповідно.

Одним із основних параметрів мережевого графіка є критичний шлях, величина якого визначає терміни виконання всього запланованого комплексу робіт. Критичний шлях – це найдовший ланцюг робіт, що веде від вихідної до завершальної події [49]. У процесі управління ходом розробки програмного забезпечення увага зосереджується на завданнях, що утворюють критичний шлях. Це дозволяє оперативно контролювати обмежене число завдань, впливати на терміни розробки, а також доцільно використовувати трудові і матеріальні ресурси. У деяких випадках в мережевому графіку може бути не один, а кілька критичних шляхів однакової тривалості [50], [51].

Серед переваг використання мережевого графіку для календарного планування робіт виділяють [50], [52]:

- можливість обґрунтовано і оперативно планувати, вибирати оптимальний варіант тривалості виконання робіт, використовувати резерви і коригувати графік в ході діяльності;
- можливість оцінити періоди, протягом яких виконання робіт може починатися і закінчуватися, а також час допустимої затримки їх виконання;
- можливість визначити, які роботи є «критичними» і тому повинні виконуватися строго за графіком, щоб розробка ПЗ була завершена в заплановані терміни;
- показує, які роботи необхідно переглянути, якщо потрібні більш стислі терміни для своєчасного закінчення процесу розробки ПЗ;
- створює основу для розрахунку потоків фінансового забезпечення процесу розробки ПЗ;
- мінімізує ризики, пов'язані з процесом розробки ПЗ.

На відміну від мережевої діаграми, що представляє послідовності виконання завдань при розробці програмного забезпечення, що розробляється, діаграма Ганта представляє собою календарний графік

розробки ПЗ, в якому зліва відображено ієрархічний перебіг виконання необхідних завдань, а справа – календар з конкретними датами [53].

При управлінні процесом розробки програмного забезпечення використовуються різноманітні засоби за допомогою яких можна здійснювати планування і управління даним процесом, організувати командну роботу тощо.

1. Microsoft Project – це система управління проектами, що допомагає менеджерам проектів при розробці планів, розподілі ресурсів, відстежуванні прогресу виконання проекту та аналізі обсягів робіт [54]. За допомогою даної програми можна побудувати критичний шлях з урахуванням ресурсів, що використовуються.

Для розрахунку критичного шляху процесу розробки ПЗ з використанням Microsoft Project необхідно задати тривалість виконання кожного завдання. Даний засіб не дозволяє автоматично визначати час, необхідний для виконання завдання, базуючись на попередньо накопичених даних або аналізуючи аналогічні завдання в інших проектах.

2. Jira – це онлайн сервіс, використовуючи який можна здійснювати планування проектом, призначати розробників на виконання певних завдань, виставляти завданням пріоритети і терміни виконання [55].

Серед переваг даного засобу виділяють можливість створення детальних звітів щодо прогресу розробки ПЗ та простоту в управлінні проектами. Але даний сервіс не дозволяє будувати мережеві діаграми та діаграми Ганта, базуючись на завданнях, що зберігаються у системі. Також не має можливості автоматично прогнозувати тривалість виконання завдання розробником певної кваліфікації.

3. Sinnaps – це засіб для управління проектами, що дозволяє отримати контроль над завданнями, графіками виконання, задіяними ресурсами та необхідними витратами [56].

Перевагами використання Sinnaps є можливість розрахунку критичних шляхів процесу розробки ПЗ, відстеження бюджету та розподілу ресурсів для

автоматичного визначення пріоритетів під час розробки ПЗ. Також даний засіб дозволяє призначати розробників на виконання завдань і повідомляє менеджерів проектів якщо члени команди перевантажені роботою.

Такі функції доступні за умови відомої тривалості виконання завдання, що визначається менеджерами проектів експертним шляхом. Sinnaps не дозволяє автоматично визначати час, необхідний на виконання завдання розробником певної кваліфікації, використовуючи попередньо створені плани розробки ПЗ.

4. WBS Schedule Pro – це сервіс для управління процесом розробки ПЗ, що містить функцію побудови мережевої діаграми для планування та управління проектами [57].

Даний сервіс дозволяє створювати мережеві діаграми та переносити їх в Microsoft Project або в інші програми управління проектами. WBS Schedule Pro автоматично розраховує критичні та некритичні шляхи, за допомогою яких можна визначити терміни завершення проекту.

Даний засіб не дозволяє автоматично визначати тривалість виконання завдання, необхідну для розрахунку критичного та некритичних шляхів.

5. Organaza – це онлайн сервіс, що використовується для планування та управління процесом розробки ПЗ. Він дозволяє розбити складну задачу на будь-яку кількість підзадач, при цьому менеджери можуть уточнити необхідні трудовитрати та встановити послідовність виконання завдань. Також даний сервіс будує діаграму Ганта з використанням доступних для реалізації проекту ресурсів в даний момент, а потім автоматично підтримує її в актуальному стані. Серед переваг даної системи також можна виділити можливість відслідковування завантаженість людських ресурсів на різних проектах та врахувати таку інформацію при побудові діаграми Ганта [58].

Даний сервіс дозволяє прогнозувати тривалість виконання підзадач, базуючись на тривалості виконання вихідної складної задачі, але за допомогою нього неможливо автоматично спрогнозувати час, необхідний на виконання початкової задачі.

6. GanttPro – це сервіс для побудови діаграм Ганта, що дозволяє планувати і управляти процесом розробки ПЗ онлайн, візуалізувати процеси, створювати завдання і призначати для їх виконання розробників, виставляти терміни виконання завдань, вказувати відсоток завершення окремих завдань і проекту в цілому [59]. До переваг даної системи відносять можливість автоматичного планування проектів; можливість вказувати власні робочі дні та години для управління проектами; автоматичний перерахунок тривалості завдань і проектів.

Але дана система не дозволяє прогнозувати тривалість виконання завдання певним розробником базуючись на інформації щодо попередньо спланованих проектів.

7. Asunote – це засіб для управління проектами, що розробляються відповідно до Agile методології. Asunote відображає фактичний прогрес у діаграмах згорання задач (burndown charts). Також дана система будує звіти для будь-якого рівня корпоративної ієрархії: від працівників до генерального директора, що дозволяє керувати окремими працівниками, командами та цілою компанією. Asunote відображає стан процесу розробки програмного забезпечення для всієї компанії та може бути інтегрована з іншими інструментами, що використовуються під час розробки ПЗ: служби Google, системи керування програмним кодом чи системи виправлення дефектів [60].

Результати порівняння існуючих засобів управління процесом розробки ПЗ наведено у табл. 1.1.

Таблиця 1.1 – Результати порівняння засобів управління процесом розробки програмного забезпечення

Засіб	Переваги	Недоліки
1	2	3
Microsoft Project	– можливість здійснювати розробку планів та розподіл ресурсів; – можливість аналізу обсягів робіт; – можливість побудови критичних шляхів.	

Продовження таблиці 1.1.

1	2	3
Jira	– можливість планування проекту; – можливість призначення розробників на виконання певних завдань; – можливість виставлення пріоритету і термінів виконання завдань; – можливість створення детальних звітів щодо прогресу розробки ПЗ.	Відсутність автоматичного визначення часу, необхідного на виконання завдання, базуючись на попередньо накопичених даних, або, аналізуючи аналогічні завдання в інших проектах
Sinnaps	– можливість контролювати завдання, задіяні ресурси та необхідні на розробку ПЗ витрати; – можливість розрахунку критичних шляхів; – можливість розподілу ресурсів для автоматичного визначення пріоритетів під час розробки ПЗ; – можливість призначення розробників на виконання завдань.	
GanttPro	– можливість виставлення термінів виконання завдань, визначення відсотку завершення окремих завдань і проекту в цілому; – можливість автоматичного планування проектів; – можливість автоматичного перерахунку тривалості завдань і проектів.	
Acunote	– можливість відображати фактичний прогрес у діаграмах згоряння задач; – можливість побудови звітів для будь-якого рівня корпоративної ієрархії; – можливість відображення стану процесу розробки ПЗ.	
WBS Schedule Pro	– можливість побудови мережевої діаграми; – можливість розрахунку критичних та некритичних шляхів.	

Продовження таблиці 1.1

1	2	3
Organaza	– можливість розбиття складної задачі на підзадачі, з встановленням послідовності їх виконання; – можливість побудови діаграми Ганта з використанням доступних ресурсів та її автоматичної підтримки в актуальному стані; – можливість відслідковування завантаженість людських ресурсів на різних проектах та врахування такої інформації при побудові діаграми Ганта.	

Отже, було проаналізовано одні з найпопулярніших засобів, що використовуються для управління процесом розробки програмного забезпечення. Такі засоби дозволяють керувати процесом розробки ПЗ, здійснювати його планування, призначати розробника на виконання завдання, але вони не дозволяють спрогнозувати тривалість виконання завдання, базуючись на накопичених даних. Такий недолік існуючих засобів можна усунути розробивши інформаційну технологію пошуку асоціативних правил щодо тривалості розробки програмного забезпечення.

1.2 Сучасні моделі розробки програмного забезпечення

Розробка програмного забезпечення – це спланований та визначений процес створення програм [61], [62]. Основними моделями розробки програмного забезпечення є [63], [64]:

1. Каскадна модель – це одна з перших моделей розробки ПЗ, що передбачає послідовне проходження усіх стадій розробки, кожна з яких повинна повністю завершитися до початку наступної [65].

Дана модель проста і зручна при керуванні проектом, тому що

розробка ПЗ проходить швидко, етапи розробки послідовні, кожен з етапів має конкретні результати. Дану модель варто застосовувати до невеликих проектів, вартість і терміни яких заздалегідь визначені.

До недоліків такої моделі належить неможливість повернутися до попереднього етапу розробки (на етапі тестування важко повернутися і змінити те, що не було передбачено на етапі аналізу). Характеристикою завдання, що враховується під час розробки ПЗ, є його пріоритет. Така модель не зручна для тривалих, складних проектів, а також для проектів, де вимоги мають середній або високий ризик зміни.

2. V – подібна модель застосовується щодо розробки ПЗ для якого особливо важливе безперебійне функціонування. Наприклад, програми в медичних установах для спостереження за станом здоров'я пацієнтів. Особливістю даної моделі є те, що на кожному з етапів розробки ПЗ відбувається його тестування: на етапі аналізу вимог здійснюється перевірка на відсутність логічних помилок в діях користувачів, розглядаються усі можливі способи використання даного ПЗ, під час кодування пишуться модульні тести і т.д [66].

Перевагами V-подібної моделі є її зрозумілість і зручність у використанні; тестування відбувається перед кодуванням, що заощаджує час у майбутньому на виправлення помилок; є зручною для використання у проектах, де вимоги зрозумілі і чітко визначені.

Недоліком даної моделі є те, що вона досить жорстка і не гнучка; ПЗ розробляється лише на етапі кодування; якщо зміни потрібно внести під час розробки ПЗ, то вимоги до ПЗ, а також тестова документація повинні бути змінені і протестовані знову. Пріоритет завдання – це характеристика, що враховується під час розробки ПЗ з використанням даної моделі.

3. Інкрементна модель передбачає кілька циклів розробки програмного забезпечення, що утворює життєвий цикл «мульти-каскад». Процедура розробки згідно даної моделі передбачає випуск на першому етапі програмного забезпечення в базовій функціональності, а потім вже

послідовне додавання нових функцій, так званих «інкрементів» [67].

Перевагами такої моделі є можливість створювати робоче програмне забезпечення швидко та на ранніх етапах його життєвого циклу; простий процес тестування та налагодження програмного забезпечення; можливість управління ризиками.

Серед недоліків виділяють: можливість виникнення проблем, пов'язаних із архітектурою системи, оскільки вимоги до програмного забезпечення не є затвердженими та можуть змінюватися в процесі його розробки; а також те, що характеристикою завдання, яка враховується під час розробки ПЗ, є лише його пріоритет.

4. Ітеративна модель передбачає створення частини функціоналу, що стає базою для визначення подальших вимог [68].

До переваг ітераційної моделі належать: можливість виявляти недоліки, пов'язані з функціональністю або проектуванням на ранніх етапах розробки; деякі функціональні можливості можуть бути розроблені на початку життєвого циклу розробки програмного забезпечення; модель легко адаптується до вимог, що змінюються; допускається паралельна розробка модулів; тестування та налагодження програмного забезпечення відбувається під час першої ітерації.

Недоліками даної моделі є необхідність у ресурсах та неможливість використання для невеликих проектів; необхідність активного управління розробкою ПЗ; не врахування інших характеристик завдань, окрім його пріоритет; визначення точної дати завершення розробки; необхідність передбачення та аналізу можливих ризиків, від чого залежить прогрес розробки.

Результати порівняльного аналізу моделей розробки програмного забезпечення наведено у табл. 1.2. Не дивлячись на те, що процес розробки ПЗ відповідно до даних моделей чітко визначений, на практиці неможливо використовувати лише одну із них.

Таблиця 1.2 – Результати порівняльного аналізу моделей розробки програмного забезпечення

Назва	Переваги	Недоліки
1	2	3
Каскадна модель	– проста і зручна при керуванні проектом; – етапи розробки послідовні, кожен з етапів має конкретні результати.	– неможливість повернутися до попереднього етапу розробки; – не зручна для тривалих, складних проектів; – врахування лише пріоритету завдання.
V – подібна модель	– тестування відбувається перед кодуванням; – зручна для використання у проектах, де вимоги зрозумілі і чітко визначені.	– жорстка і не гнучка; – ПЗ розробляється лише на етапі кодування; – повторне тестування, якщо зміни потрібно внести в середині процесу розробки ПЗ; – при розробці ПЗ враховується лише пріоритет завдання.
Інкрементна модель	– можливість створювати програмне забезпечення на ранніх етапах його життєвого циклу; – простий процес тестування програмного забезпечення; – можливість управління ризиками.	– можливість виникнення проблем, пов'язаних із архітектурою системи; – врахування лише пріоритету завдання.
Ітеративна модель	– можливість виявляти недоліки на ранніх етапах розробки; – легко адаптується до вимог проекту; – допускається паралельна розробка модулів; – тестування програмного забезпечення відбувається під час першої ітерації.	– не підходить для невеликих проектів; – необхідність активного управління розробкою ПЗ; – неможливість визначення дати завершення розробки; – враховується лише пріоритет завдання під час розробки ПЗ; – необхідність передбачення та аналізу можливих ризиків.

Як видно із табл. 1.2 при розробці програмного забезпечення, з використанням розглянутих моделей, враховується лише пріоритет завдання, в той час як такі характеристики завдання як складність, критичність та тривалість його виконання не беруться до уваги.

Для кожної з вищезгаданих моделей процесу розробки ПЗ, основними етапами є ті, що зображені на рис. 1.2 [69] – [72]:

1. Аналіз. На даному етапі відбувається дослідження предметної області і виявляються важливі вимоги до майбутнього програмного забезпечення з точки зору замовника або користувача.

Результатом є специфікація на програмне забезпечення, що розробляється.

2. Проектування. Під час проектування вимоги користувача до програмного забезпечення перетворюються в детальні і конкретні вимоги до внутрішньої будови та функціонування майбутнього програмного забезпечення з точки зору розробників.

Результат – прототип розроблюваного програмного забезпечення.

3. Програмування (кодування) – це процес реалізації програмного забезпечення з використанням мов програмування та відповідних засобів розробки.

Результат – програмне забезпечення, готове до використання.

4. Тестування та виправлення помилок – процес виявлення та усунення помилок у розробленому програмному забезпеченні і встановлення відповідності створеного ПЗ до його специфікації.

Результатом є налагоджене програмне забезпечення, що відповідає специфікації та може бути передане замовнику.

5. Документування. Під час даного процесу створюється документація, що складається з опису програмного забезпечення як з точки зору його створення, так і з точки зору його використання.

Кінцевий результат – документація на розроблену програму.



Рисунок 1.2 – Основні етапи розробки програмного забезпечення

При розробці програмного забезпечення може виникати необхідність у інших етапах, але вони є не основними та залежать від конкретно обраної моделі розробки.

Отже, було проаналізовано сучасні моделі процесу розробки програмного забезпечення та встановлено, що під час їх використання враховується лише така характеристика завдання як пріоритет. Тому існує необхідність у розробці інформаційної моделі, яка б враховувала інші характеристики завдань також, а саме: складність, критичність та тривалість виконання. Також було визначено, що на практиці під час створення ПЗ зазвичай використовується декілька моделей, що мають встановлені етапи, тривалість розробки яких можна визначати за допомогою використання методів інтелектуального аналізу даних щодо розробки програмного забезпечення.

1.3 Особливості використання методів інтелектуального аналізу даних щодо розробки програмного забезпечення

Інформація, що накопичується під час розробки програмного забезпечення, зберігається компаніями-розробниками з метою її подальшого аналізу. Сукупність інформації, що зберігається в репозитаріях, може зіграти

ключову роль у покращенні процесу розробки ПЗ та якості розроблюваних програм [73].

Технологія Data Mining (DM) дозволяє аналізувати та отримувати корисні для експерта закономірності з інформації. Знайдені шаблони знань можуть допомогти розробникам ПЗ у передбаченні, плануванні та розумінні аспектів процесу розробки програми і можуть підтримувати майбутню розробку і управління проектом [74] – [77].

Основними методами ІАД, що можуть використовуватися щодо розробки програмного забезпечення, є [78] – [87]:

1. Інтелектуальний аналіз текстової інформації (листи, коментарі, документація), що дозволить визначати активних учасників розробки програмного забезпечення.

2. Кластеризація, що може використовуватися для визначення груп схожих модулів програмного забезпечення, базуючись на кількості зроблених модифікацій; або для групування знайдених дефектів ПЗ відповідно до конкретних компонентів, що дозволить ефективно їх виправляти.

3. Класифікація, що може допомогти у визначенні механізму ідентифікації вразливих модулів програмного забезпечення на основі атрибутів модулів чи системи.

4. Добування частих шаблонів або асоціативних правил. Аналіз системних помилок, що виникли при роботі програмних модулів, може винайти взаємозв'язки між ними, базуючись на характеристичних особливостях модулів і категоріях помилок. Також даний метод ІАД можна застосовувати для прогнозування розробника, який виправлятиме конкретні помилки в ПЗ, що скоротить час, необхідний для підбору людських ресурсів, потрібних для виправлення дефектів.

5. Міркування на основі прецедентів (Case-Based Reasoning – CBR), що дозволяє знаходити прецеденти в минулому, аналізувати їх та застосовувати ці знання до нової проблеми. Наприклад, при тестуванні ПЗ (після його

поділу на компоненти) використовується CBR для знаходження необхідного набору тест-кейсів для конкретного компоненту базуючись на даних, отриманих при тестуванні схожих компонентів раніше.

На кожному із етапів розробки ПЗ можна використовувати відповідні методи інтелектуального аналізу даних, як представлено на рис. 1.3.

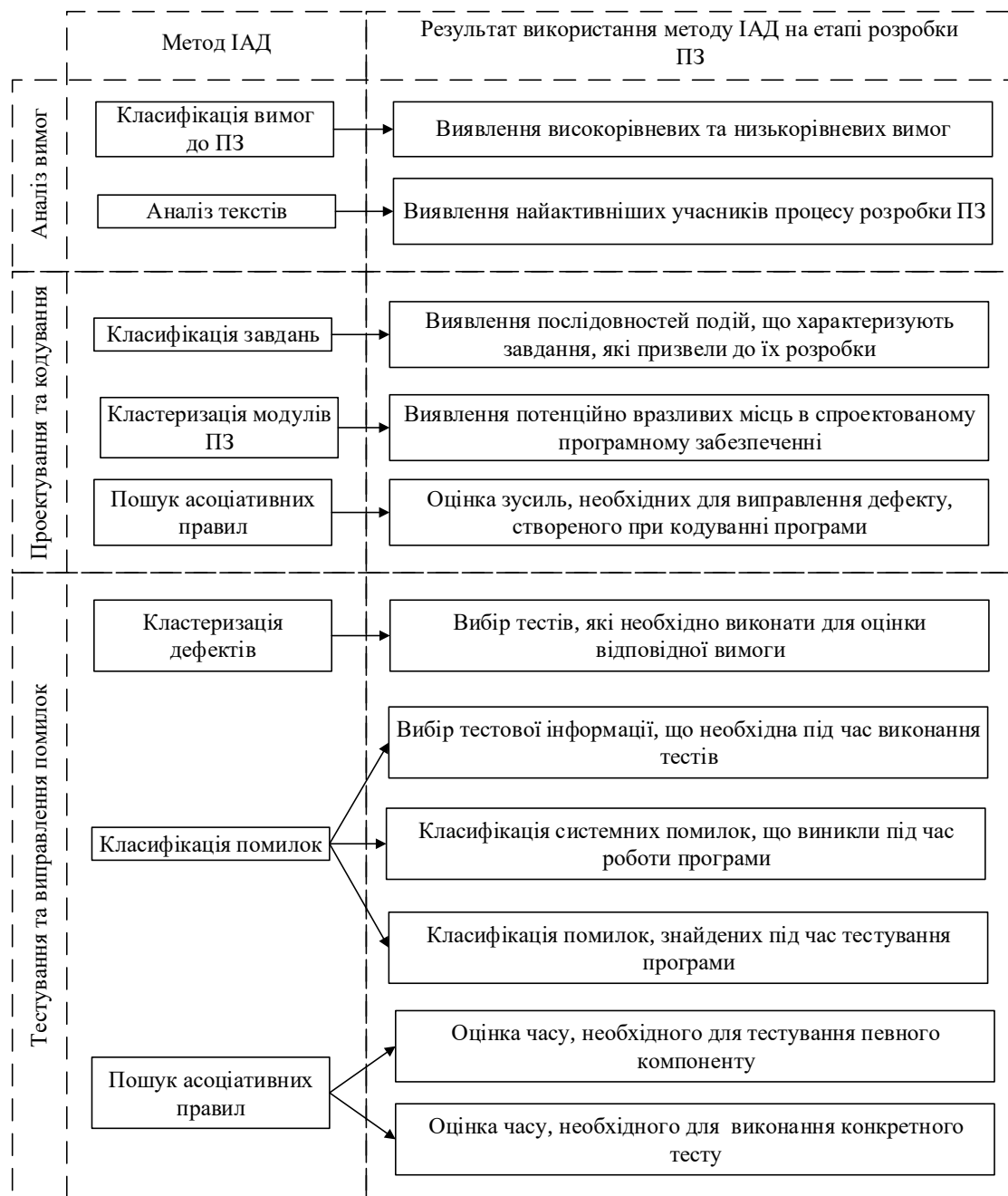


Рисунок 1.3 – Методи ІАД, що можна використовувати на етапах розробки програмного забезпечення

Для представлення процесу розробки ПЗ можна використовувати різні формальні підходи.

В роботі [88] пропонується формалізація варіантів використання програмного забезпечення за допомогою моделі Крипке. Автори застосовують шаблон, що здійснює перетворювання варіантів використання ПП в структуру Крипке [89]. В подальшому дана структура може використовуватися для формальної верифікації системи [90] або для автоматичного визначення тестових прикладів.

Результатом такого підходу є усунення розриву між неформальним описом вимог і формальною специфікацією, що полегшує роботу користувачів.

Запропонований підхід доцільно використовувати на етапі підготовки даних для тестування та на етапі тестування ПЗ [91]. Проте його не можна застосувати, наприклад, при проектуванні ПЗ чи його програмуванні.

У роботі [92] автори пропонують формалізувати варіанти використання ПЗ за допомогою Х-машин. Така формалізація дозволить отримати повний набір тестів, необхідних для верифікації програмного забезпечення.

Вищезгаданий формальний підхід перетворення варіантів використання програмного забезпечення [93] у різні структури доцільно використовувати на етапі тестування програмного забезпечення. Але за допомогою нього неможливо представити розробку ПЗ на інших її етапах.

У роботі [94] пропонується проектувати ПЗ з використанням UML діаграм і мереж Петрі, що дозволить знаходити і усувати логічні помилки (зациклення, кінцеві маркування, мертві переходи). Створення програмного забезпечення на основі спільного використання UML діаграм і мереж Петрі [95] поділяють на етапи: вибір і розробка необхідних UML діаграм; перетворення отриманих діаграм в мережі Петрі; аналіз мереж Петрі; внесення необхідних змін до UML діаграми відповідно до результатів аналізу.

Такі перетворення доцільно використовувати при проектуванні програмного забезпечення, але його не можна використовувати при розгляді процесу розробки ПЗ на різних рівнях деталізації.

Автори [96] розглядають MDD модель процесу розробки програмного забезпечення, що здійснює перетворення діаграми послідовностей у мережі Петрі.

Автори пропонують здійснювати розбиття діаграми послідовностей на блоки і відображати їх за допомогою мереж Петрі. Для створення загальної мережі Петрі такі блоки об'єднуються, в результаті чого аналіз програмного забезпечення, що розробляється, не становить жодних складностей.

У роботі [97] пропонується використання підходу, що базується на методі Монте-Карло для перевірки надійності програмного забезпечення. Під час нього використовуються часті набори даних для визначення властивостей заданого об'єкта. Результати дослідження відображають відсоток надійності програмного забезпечення.

Такий підхід доцільно використовувати при тестуванні ПЗ, але за допомогою нього не можна промодельовати усі етапи процесу розробки ПЗ.

У табл. 1.3 наведено порівняльну характеристику розглянутих вище підходів та вказано етапи розробки ПЗ на яких дані підходи доцільно застосовувати.

Таблиця 1.3 – Порівняльна характеристика формальних підходів представлення процесу розробки програмного забезпечення

Формальний підхід	Етап процесу розробки ПЗ	Недолік формального підходу
Структура Крипке	Етап підготовки тестових даних Етап тестування ПЗ	Неможливо використовувати підхід для представлення процесу розробки ПЗ на різних рівнях деталізації
Х-машини	Етап тестування ПЗ	
Мережі Петрі та UML діаграма	Етап проектування ПЗ	
Підхід Model Driven Development	Етап проектування ПЗ	
Метод Монте-Карло	Етап тестування ПЗ	

Як видно з табл. 1.3 на даний момент переважають підходи, які відображають розробку програмного забезпечення лише на певному етапі його розробки і не можуть бути застосовані на різних рівнях деталізації.

Отже, проведено аналіз методів інтелектуального аналізу даних, що використовуються під час розробки програмного забезпечення, та зроблено висновки про необхідність їх використання під час даного процесу, що дозволить автоматизувати певні його етапи. Також досліджено актуальність вирішення задачі представлення процесу розробки ПЗ на різних рівнях деталізації за допомогою формальних підходів.

1.4 Обґрунтування доцільності пошуку і використання асоціативних правил щодо тривалості розробки програмного забезпечення

Пошук асоціативних правил щодо тривалості розробки програмного забезпечення є актуальною задачею, що може бути застосована для удосконалення даного процесу на різних його етапах.

Задача пошуку асоціативних правил щодо тривалості розробки програмного забезпечення може бути сформована наступним чином.

Вхідною множиною даних є набір $Q = \{Q_1, Q_2 \dots Q_i \dots Q_n\}$, де $i = \overline{1, n}$, n – кількість вхідних даних; а Q_i – це множина:

$$Q_i = \{T_i, D_j, t_{ij}\}, \quad (1.1)$$

де D_j – це j -й розробник, T_i – це i -е завдання, що має ряд характеристик і виконане j -м розробником, а t_{ij} – час, необхідний на розв’язання i -го завдання j -м розробником.

Задача пошуку асоціативних правил щодо тривалості розробки програмного забезпечення зводиться до пошуку асоціативних правил, що

відображені у виразі:

$$T_i \cap D_j \rightarrow t_{ij}, T_i \cap D_j \cap t_{ij} \rightarrow \emptyset. \quad (1.2)$$

Тобто, необхідно знайти асоціативні правила виду: «якщо завдання T_i буде розв'язуватися розробником D_j , то на його виконання буде витрачено t_{ij} часу».

Асоціативні правила – це закономірності типу «із події X , слідує подія Y », тобто $X \rightarrow Y$, для яких обов'язково повинна виконуватися умова, що $X \cap Y = \emptyset$. Залежно від задачі, яку потрібно вирішити, використовується відповідний вид асоціативних правил [98] – [108]:

- позитивні $X \rightarrow Y$;
- негативні $X \rightarrow \bar{Y}$ або $\bar{X} \rightarrow Y$. Пошук негативних правил разом з позитивними обумовлений необхідністю детального опису залежностей, що досліджуються. Такий підхід допоможе здійснювати більш точне прогнозування, використовуючи отримані правила;
- узагальнені: пошук асоціативних правил здійснюється не лише між елементами даних, а й між елементом і групою даних; між групами даних. Такий пошук доцільно здійснювати, наприклад, для аналізу зв'язку між симптомами хвороби при медичному діагностуванні;
- часові: циклічні АП та АП, що описують залежності, зв'язані з деякими інтервалами часу. Пошук таких асоціативних правил доцільно здійснювати при роботі з БД, у яких збережені події, пов'язані у часі.

Будь-яке асоціативне правило може бути охарактеризоване двома числовими величинами: підтримкою асоціативного правила та його достовірністю [109].

Підтримкою $supp(X \rightarrow Y)$ асоціативного правила $X \rightarrow Y$ є величина, що дорівнює відношенню кількості записів $X \cup Y$ в БД [109], [110].

$$\text{supp}(X \rightarrow Y) = \text{supp}(X \cup Y). \quad (1.3)$$

Достовірністю $\text{conf}(X \rightarrow Y)$ асоціативного правила $X \rightarrow Y$ є величина, що дорівнює відношенню його підтримки $\text{supp}(X \rightarrow Y)$ до підтримки $\text{supp}(X)$ множини подій X [109], [111].

$$\text{conf}(X \rightarrow Y) = \frac{\text{supp}(X \rightarrow Y)}{\text{supp}(X)}. \quad (1.4)$$

Для визначення значення мінімальної підтримки minsupp можна використовувати функцію, відображену у виразі (1.5) [112]. Вперше використання цієї функції для пошуку частих предметних наборів було запропоновано в [113]. Це дозволяє встановити високе значення мінімальної підтримки, коли даних в БД небагато, а потім зменшувати його у випадку збільшення кількості даних:

$$\text{minsupp}(h) = e^{-\alpha h - \beta} + \gamma, \quad (1.5)$$

де h – це кількість записів в БД;

α, β, γ – додатні константи. Константа γ – це мінімальне значення, якому може дорівнювати достовірність conf знайдених АП. Константи α та β впливають на те, як різко повинна зменшуватися $\text{minsupp}(h)$, коли h збільшиться.

При встановленій високій мінімальній підтримці може бути визначено лише кілька правил, але багато часу буде витрачено на сканування бази даних. Вибір низького значення мінімальної підтримки може призвести до надлишкових правил. Тому виникає необхідність у виборі оптимального значення для даних показників [114].

Задачу пошуку асоціативних правил щодо тривалості розробки ПЗ

можна розділити на дві підзадачі:

1. Підзадачу визначення частих предметних наборів для пошуку АП щодо тривалості розробки програмного забезпечення, в яких рівень підтримки не нижче заданого експертом порогового значення $\text{minsupp}(X)$.

2. Підзадачу пошуку правил $(X \rightarrow Y)$ щодо тривалості розробки програмного забезпечення, що мають рівень достовірності не нижче заданого експертом порогового значення $\text{minconf}(X \rightarrow Y)$.

Для підзадачі визначення частих предметних наборів використовують методи, що відображені на рис. 1.4 і які можуть бути класифіковані за способом цього визначення [115] – [121]:

- методи, що визначають кандидатів для частих предметних наборів;
- методи, що не здійснюють визначення кандидатів перед пошуком частих предметних наборів.

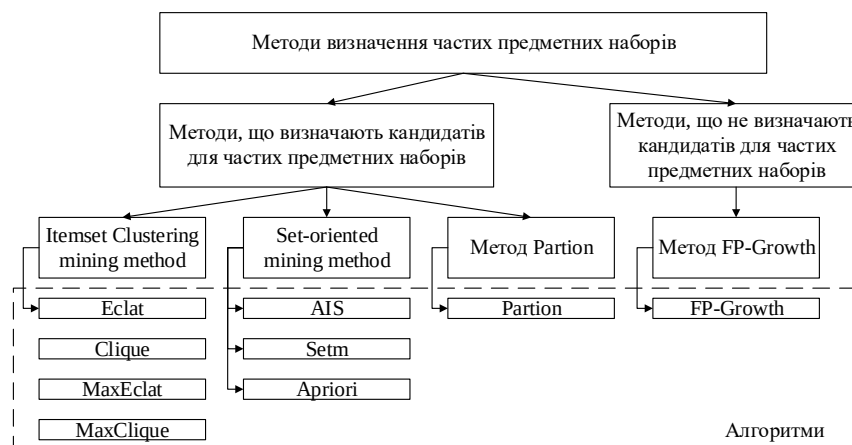


Рисунок 1.4 – Методи визначення частих предметних наборів

Розглянемо кожен метод детально та здійснимо їх порівняльний аналіз.

Метод *Set-oriented mining* був запропонований для визначення частих предметних наборів. З його використанням створена велика кількість алгоритмів, таких як AIS, SETM, Apriori тощо [122], [123].

Обмеження: вказаний метод обробляє інформацію лише в статичних

базах даних.

Переваги: можна застосовувати до потужних БД; метод простий для розуміння і може бути використаний для створення нових алгоритмів визначення частих предметних наборів.

Недоліки: процес визначення можливих частих предметних наборів тривалий та використовує пам'ять комп'ютера; виконується багаторазове сканування БД; можливість роботи лише зі статичними даними.

Опис методу. На першому кроці відбувається підрахунок кількості транзакцій, в яких міститься одиничний елемент it_1 . Транзакцією називається набір даних, що містить елементи it . Якщо вимога щодо мінімального значення підтримки виконується, то результат зберігається в таблиці Tb_1 , що має структуру $Tb_1(it_1, \Sigma)$, де Σ – це кількість транзакцій в яких міститься одиничний елемент it_a .

На другому етапі відбувається підрахунок кількості транзакцій в яких міститься комбінація з двох елементів, що присутні в Tb_1 . Якщо вимога щодо мінімального значення підтримки виконується для цієї комбінації, то результат зберігається в таблиці $Tb_2(it_1, it_2, \Sigma)$ і є частим набором з двох елементів.

З кожним наступним етапом відбувається підрахунок кількості транзакцій, в яких містяться комбінації з більшою кількістю елементів (+ 1 елемент на кожному новому етапі). Розглядаються лише ті елементи, які записані в Tb_{c-1} , що є таблицею з попереднього етапу, оскільки вони задовольняють вимогу мінімальної підтримки.

Таблиця з частими предметними наборами має вигляд $Tb_c(it_1, it_2, \dots, it_c, \Sigma)$. Пошук частих предметних наборів припиняється коли $Tb_{c+1} = \emptyset$.

На базі даного методу розроблена велика кількість алгоритмів, деякі з них наведено нижче.

Алгоритм *AIS* використовує процедуру оцінки для визначення об'єктів, які необхідно враховувати під час проходження по БД. Дана процедура забезпечує баланс між кількістю проходжень по БД і кількістю предметних наборів, що утворюються в результаті цього проходження [124].

В даному алгоритмі використовується технологія скорочення для уникнення використання предметних наборів, які не є частими предметними наборами.

Алгоритм *SETM* був створений з метою використання SQL запитів для пошуку частих предметних наборів у потужних БД. Кроки алгоритму складаються із чітко визначених і простих операцій. Використання мови-запитів SQL дозволяє обробляти БД за допомогою вбудованого інструментарію [125].

Алгоритм *Apriori* [126] найчастіше використовується для пошуку асоціативних правил, тому існує значна кількість його модифікацій. Хоча даний алгоритм визначає велику кількість предметних наборів при першому скануванні БД, але при наступному проходженні по БД вона зменшується і більшість виявлених предметних наборів є частими. Швидкість обробки даних у БД при його використанні значно вища ніж у *AIS* чи *SETM*.

Method Partition [127] дозволяє здійснювати пошук асоціативних правил у потужних БД. Основною його відмінністю від Set-oriented mining method є те, що перед пошуком частих предметних наборів БД розділяється на декілька частин кожна з яких обробляється окремо. Після визначення значень підтримки частих предметних наборів з кожної частин здійснюється ще один прохід по БД щоб перевірити, що локальні часті предметні набори також є і глобальними.

Обмеження: дані в БД зберігаються відсортованими в лексикографічному порядку у форматі $\langle ID, it \rangle$, де *ID* – це унікальний ідентифікатор транзакції, що розглядається, *it* – окремий елемент, з яких складається транзакція.

Також припускається, що розміри БД відомі, а значення *ID* монотонно

зростають.

Переваги: метод дозволяє швидко здійснювати пошук у потужних БД; відбувається лише два сканування БД для визначення частих предметних наборів, що зменшує використання фізичної пам'яті комп'ютера.

Недоліки: виконання даного методу потрібно здійснювати на комп'ютері з великою оперативною пам'яттю; недосконалий підхід поділу БД на частини на першому етапі.

Опис методу. На першому етапі відбувається розділення БД на частини, що не мають спільних даних між собою. Для визначення кількості частин використовують евристичні методи, з урахуванням того, що їх розміри мають дозволяти обробити їх в оперативній пам'яті комп'ютера.

На цьому етапі визначаються усі можливі локальні часті предметні набори, які потім об'єднуються для отримання потенційних глобальних частих предметних наборів.

На другому етапі відбувається підрахунок значення підтримки для глобальних частих наборів, що, в подальшому, будуть використанні для пошуку АП.

У методі *Itemset Clustering mining method* [126] визначаються потенційні максимальні часті предметні набори, що складаються як мінімум із 2 елементів. Для цього використовуються такі методи кластеризації [121]: кластеризація на основі класів еквівалентності, maximal Hypergraph Clique кластеризація.

Для визначення кінцевих частих предметних наборів використовуються такі методи проходження по графу: знизу доверху та гібридний (знизу доверху та зверху донизу).

Залежно від способу кластеризації та проходження по графу розрізняють такі алгоритми пошуку асоціативних правил [121]:

1. Eclat: використовуються класи еквівалентності та проходження «знизу доверху».
2. MaxEclat: використовуються класи еквівалентності та гібридний

метод проходження по графу.

3. Clique: використовуються *maximal hypergraph clique* та проходження «знизу доверху».

4. MaxClique: використовуються *maximal hypergraph clique* та гібридний метод проходження по графу.

Обмеження: дані в БД повинні бути представлені у вертикальному вигляді, що дозволить їх кластеризувати за одне проходження по БД.

Переваги: використовується мало оперативної пам'яті комп'ютера; не потрібно будувати складну хеш-структуру для визначення частих предметних наборів; відбувається лише одне сканування БД.

Недоліки: необхідно здійснити вибір параметрів для алгоритму кластеризації.

До методів, що не здійснюють визначення кандидатів для частих предметних наборів належить *Frequent Pattern Growth (FP-Growth)*.

Особливістю методу ***FP-Growth*** [116], [128] є те, що у ньому використовується технологія побудови алгоритмів – «поділу і захоплення» (*divide and conquer*), що дозволяє виконати декомпозицію однієї складної задачі на безліч більш простих.

Опис методу. На першому етапі відбувається перетворення БД в деревовидну структуру, що називається FP-дерево (*Frequent-Pattern Tree*). Під час цього процесу також підраховується значення підтримки для кожного елемента.

На другому етапі здійснюється добування частих предметних наборів із FP-дерева.

Обмеження: FP – розмір дерева не може перевищувати розмір пам'яті комп'ютера, на якому виконується його створення.

Переваги: розмір FP-дерева досить малий, що дозволяє уникнути затратної процедури визначення кандидатів для частих предметних наборів; швидкість пошуку частих предметних наборів вища, ніж у попередніх методах.

Недоліки: неможливо побудувати FP-дерево, що матиме розмір більший, ніж пам'ять комп'ютера; хоча FP-дерево досить компактне, але для його побудови потрібно двічі сканувати БД, що може являти собою нетривіальні витрати; дозволяє обробляти лише статичні бази даних.

Результати порівняльного аналізу методів визначення частих предметних наборів для пошуку асоціативних правил щодо тривалості розробки програмного забезпечення наведено в табл. 1.4.

Таблиця 1.4 – Результати порівняльного аналізу методів визначення частих предметних наборів для пошуку асоціативних правил щодо тривалості розробки ПЗ

Метод	Переваги	Недоліки
1	2	3
Set-oriented	– можна застосовувати до потужних БД; – метод простий для розуміння.	– процес визначення частих предметних наборів тривалий та використовує пам'ять комп'ютера; – виконується багаторазове сканування БД; – можливість роботи лише зі статичними даними.
Partition	– швидкий пошук у потужних БД; – лише два сканування БД.	– необхідність у комп'ютері з потужною оперативною пам'яттю; – недосконалий підхід поділу БД на частини на першому етапі.
Itemset Clustering	– використання невеликої кількості оперативної пам'яті; – немає необхідності у побудові хеш-структури; – лише одне сканування БД.	– необхідність здійснювати вибір параметрів для алгоритму кластеризації.
FP Growth	– розмір FP-дерева досить малий; – висока швидкість пошуку частих предметних наборів.	– розмір FP-дерева не може бути більшим, ніж пам'ять комп'ютера; – два сканування БД, що може призвести до нетривіальних витрат; – обробка лише статичних даних.

Для вирішення підзадачі пошуку асоціативних правил $X \rightarrow Y$ використовується один загальний підхід, який полягає в наступному [116]:

1. Для кожного частого набору довжиною w елементів усі можливі комбінації з $w - 1$ елементів розглядаються як умова (X) ; а елемент, що залишився - як наслідок (Y) .

2. Для кожної комбінації X та Y перевіряється чи значення достовірності дорівнює або більше значення заданої мінімальної достовірності $minconf(X \rightarrow Y)$.

3. Якщо умова мінімальної достовірності виконується, асоціативне правило записується у вигляді $X \rightarrow Y$.

Аналіз методів визначення частих предметних наборів, що можуть бути використані для пошуку асоціативних правил щодо тривалості розробки програмного забезпечення показав, що варто застосовувати метод FP – Growth, оскільки він відповідає наступним вимогам: дозволяє обробляти потужні БД; швидкість визначення частих предметних наборів висока.

Для усунення існуючих недоліків даного методу, завдання, серед яких здійснюватиметься пошук АП, необхідно розділити на три класи. Для такої класифікації варто застосовувати методи класифікації даних, метою яких є точне визначення цільового класу даних.

Класифікація – це двоетапний процес [129] – [131]:

1. На першому етапі на наборі навчальних даних створюється модель шляхом застосування алгоритмів класифікації.

2. На другому етапі розроблена модель тестується із використанням попередньо визначеного набору тестів для вимірювання точності розробленої моделі.

Серед основних методів класифікації даних, які можуть бути застосовані для модифікації методу пошуку асоціативних правил щодо тривалості розробки програмного забезпечення, можна виділити такі: дерева рішень, метод опорних векторів (SVM), метод К-найближчих сусідів та

наївний Байєсівський метод.

Дерева рішень

Основна ідея даного методу полягає у рекурсивному розбитті множини об'єктів із навчальної вибірки на підмножини, що складаються з об'єктів, які належать певним класам [132].

Переваги: правила для класифікації створюються з навчальних даних; швидка побудова короткого дерева рішень; розглядається увесь набір даних для створення дерева; побудовану модель можна легко інтерпретувати; простий в реалізації; можна використовувати як дискретні, так і неперервні величини [133].

Недоліки: на етапі побудови дерев рішень неможливо сказати чи вибрана початкова змінна дасть оптимальний варіант розбиття; існує проблема зупинки розбиття дерева; класифікація неперервних даних може бути затратною: при використанні недостатнього тренувального набору класифікація не точна [133].

Метод *опорних векторів (SVM)* базується на принципі мінімізації структурного ризику, метою якого є визначення місця розташування меж рішень (гіперплощин), які забезпечують оптимальне розділення даних. Якісна класифікація даних досягається за рахунок гіперплощини, яка має найбільшу відстань до найближчої точки навчальної вибірки будь-якого класу, оскільки в загальному випадку чим більша відстань, тим менша помилка класифікатора [134].

Переваги SVM: точніший метод, ніж дерева рішень [135].

Недоліки SVM: потребує затрат часу на великі матричні операції; витрати часу на тренування SVM класифікаторів зростають квадратично, залежно від кількості прикладів, тому дослідження постійно прагнуть до ефективного алгоритму навчання [135].

У методі *k-найближчих сусідів* [136] кожна ознака належить до переважного класу найближчих сусідів, де k -параметр методу. Згідно з гіпотезою компактності [137], тестова ознака матиме таку ж мітку, як і

навчальні ознаки в локальній множині, що оточує їх.

Оскільки при застосуванні даного методу зразки даних необхідно зберігати в пам'яті під час виконання алгоритму, то даний метод вважається методом на основі пам'яті. Для того щоб уникнути обмеження в пам'яті, зменшують розмір набору даних: шаблони даних, що повторюються та не містять додаткових даних виключаються з навчальної вибірки даних [137].

Узагальнені кроки алгоритму в основі якого лежить даний метод [137]:

1. Визначення значення параметра k – кількість найближчих сусідів.
2. Для кожного об'єкта v в тестовій множині визначається відстань $\delta(v, \omega)$ між v і кожним об'єктом ω із навчальної вибірки.
3. Об'єкт ω відносить до класу, до якого найменша відстань.

Переваги методу k -найближчих сусідів: відсутній навчальний процес; не потрібно робити жодних припущень про характеристики об'єктів; комплексні поняття можуть бути вивчені локальним наближенням, використовуючи прості процедури [138].

Недоліки методу k -найближчих сусідів: обчислювальна складність методу висока; не можна інтерпретувати модель; важко знайти k -найближчих сусідів, коли набір даних потужний; продуктивність методу залежить від розмірів вибірки [138].

Наївний Байєсівський метод базується на теоремі Байєса і його доцільно застосовувати, коли на вхід подається велика кількість даних. При використанні наївного класифікатора Байєса вважається, що присутність (або відсутність) певної ознаки (атрибута) класу не має відношення до присутності (або відсутності) будь-якої іншої ознаки при наданні змінної класу [139].

До переваг даного методу належать: швидкість класифікації даних; можливість створювати імовірнісні прогнози [140].

Серед недоліків даного методу виділяють: відсутність у можливості реалізації регресії; наявність багатьох припущень [140].

Результати порівняння зазначених методів класифікації даних, які

можуть бути застосовані для модифікації методу FP-Growth пошуку асоціативних правил щодо тривалості розробки програмного забезпечення, наведено в табл. 1.5.

Таблиця 1.5 – Результати порівняння методів класифікації даних, які можуть бути застосовані для удосконалення FP-Growth методу

Назва методу	Переваги	Недоліки
1	2	3
Дерева рішень	– швидка побудова короткого дерева рішень; – розглядається увесь набір даних для створення дерева; – побудовану модель, можна легко інтерпретувати; – простий в реалізації; – можна використовувати як дискретні, так і неперервні величини.	– неможливо сказати чи вибрана початкова змінна дасть оптимальний варіант розбиття; – існує проблема зупинки розбиття дерева; – класифікація неперервних даних може бути затратною; – при використанні недостатнього тренувального набору, класифікація не точна.
k-найближчих сусідів	– відсутній навчальний процес; – не потрібно робити жодних припущень про характеристики об'єктів; – комплексні поняття можуть бути вивчені локальним наближенням.	– обчислювальна складність методу висока; – не можна інтерпретувати модель; – важко знайти k-найближчих сусідів, коли набір даних потужний; – продуктивність методу залежить від розмірів вибірки.
Наївний Байєсівський метод	– класифікація даних відбувається швидко; – існує можливість створювати імовірнісні прогнози.	– відсутність можливості реалізації регресії; – наявність багатьох припущень.
Опорні вектори	– точніший ніж дерева рішень.	– потребує затрат часу на великі матричні операції.

Як видно із табл. 1.5 кожен із методів може бути застосований для модифікації методу визначення частих предметних наборів для пошуку асоціативних правил щодо тривалості розробки програмного забезпечення.

Але, оскільки навчальна вибірка є достатньою та потрібно отримати легко інтерпретовані дані за короткий термін, то для цього було обрано метод дерева рішень.

Як зазначалось раніше, використання методів Data Mining можливе на усіх етапах розробки програмного забезпечення.

Так, в роботі [141] пропонується пошук і використання асоціативних правил для визначення порушень в архітектурі ПЗ. Першим кроком виконання даного підходу є добування усіх версій комп'ютерного коду програми з репозитарію. Наступним кроком є отримання усіх залежностей між модулями даного програмного забезпечення. Потім, використовуючи запити мови Prolog, записи в БД перетворюються в набір даних, що може бути використаний FP-Growth-алгоритмом для пошуку залежностей в архітектурі та структурі ПЗ. Останнім кроком є пошук залежностей, які свідчать про порушення в архітектурі ПЗ.

Такий підхід допоможе визначати вразливі місця в архітектурі ПЗ на ранніх етапах його розробки, що дозволить уникнути додаткових матеріальних затрат на їх виправлення в майбутньому.

Підхід, запропонований у роботі [142], дозволить менеджерам проекту визначати кількість необхідних ресурсів для розробки ПЗ та ефективно ними управляти. Автори пропонують застосовувати теорію нечітких множин та пошук асоціативних правил з використанням алгоритму Apriori для перерахунку необхідних людських ресурсів для розробки ПЗ.

Запропонований підхід, базуючись на початковій оцінці необхідних ресурсів для розробки ПЗ, дозволить в процесі розробки визначати необхідні ресурси для його реалізації. Це надасть можливість менеджерам проекту ефективно планувати діяльність кожного учасника даного процесу.

У роботі [10] автори розглядають можливість використання технології пошуку асоціативних правил для визначення розробника, якого буде призначено для виправлення знайденого дефекту у програмі. У своєму дослідженні автори використовують алгоритм Apriori та такі характеристики дефекту як: важливість, пріоритетність, короткий опис та відповідальна

особа за дефект. Знайдені асоціативні залежності сприяють визначенню розробника, якого можна призначити на виправлення іншого дефекту з такими ж параметрами.

Даний підхід дозволить автоматизувати процес призначення відповідальної особи за виправлення конкретного дефекту, та може бути використаний менеджерами проекту при плануванні діяльності учасників команди розробки ПЗ.

Отже, формалізовано задачу пошук асоціативних правил щодо тривалості розробки програмного забезпечення, що є актуальною та потребує вирішення. Проведено аналіз існуючих методів визначення частих предметних наборів для пошуку асоціативних правил щодо тривалості розробки програмного забезпечення. З'ясовано, що методом, який доцільно застосовувати щодо розробки програмного забезпечення є FP-Growth, але для усунення зазначених недоліків, його потрібно удосконалити. Удосконалення полягає у класифікації завдань перед визначенням частих предметних наборів для пошуку асоціативних правил. Це дозволить скоротити тривалість знаходження таких правил та підвищити точність визначення часу, необхідного на виконання завдання розробником певної кваліфікації за рахунок їх використання.

1.5 Постановка задачі

Інформаційна технологія пошуку асоціативних правил щодо тривалості розробки програмного забезпечення дозволить усунути такий недолік існуючих засобів, що використовуються під час розробки ПЗ, як відсутність можливості визначення тривалості виконання завдання, базуючись на накопичених даних. Як показав аналіз моделей процесу розробки програмного забезпечення, визначення тривалості його етапів можна здійснити з використанням методів інтелектуального аналізу даних, що дозволить їх автоматизувати. Проведений аналіз існуючих методів визначення частих предметних наборів довів необхідність усунення недоліків

методу FP-Growth за рахунок попередньої класифікації завдань, серед яких здійснюватиметься пошук асоціативних правил. Таке удосконалення дозволить підвищити точність визначення тривалості виконання завдання розробником, а також скоротити тривалість пошуку асоціативних правил, які для цього використовуються.

Задача дослідження полягає у пошуку і використанні асоціативних правил щодо розробки програмного забезпечення для подальшого їх застосування менеджером проекту при плануванні процесу розробки програмного забезпечення.

Для досягнення цієї мети необхідно розв'язати такі задачі:

- розробити інформаційну модель процесу пошуку асоціативних правил щодо тривалості розробки програмного забезпечення;
- удосконалити метод FP-Growth визначення частих предметних наборів для пошуку асоціативних правил щодо тривалості розробки ПЗ ;
- розробити інформаційну технологію процесу пошуку асоціативних правил щодо тривалості розробки програмного забезпечення;
- розробити інформаційну систему пошуку асоціативних правил щодо тривалості розробки програмного забезпечення;
- дослідити результати впровадження інформаційної системи пошуку асоціативних правил щодо тривалості розробки програмного забезпечення.

1.6 Висновки до розділу 1

В результаті проведеного аналізу засобів, що використовуються для управління процесом розробки програмного забезпечення, зроблено висновок, що вони не дозволяють спрогнозувати тривалість виконання окремого завдання та проекту в цілому, використовуючи дані про вже реалізовані завдання. Тому існує необхідність у розробці інформаційної системи, що дозволить здійснювати таке визначення, за рахунок використання інформаційної технології пошуку асоціативних правил щодо тривалості розробки програмного забезпечення.

Здійснений аналіз основних моделей процесу розробки програмного

забезпечення встановив, що під час їх використання враховується лише пріоритет завдання, а тому існує необхідність у розробці інформаційної моделі, яка враховує також такі характеристики завдань як: складність, пріоритет, критичність та тривалість його виконання. Проведений аналіз також показав, що основні етапи процесу розробки програмного забезпечення є однаковими для розглянутих моделей і тривалість їх реалізації може бути прогнозованою за рахунок використання методів інтелектуального аналізу даних.

Проведений аналіз методів інтелектуального аналізу даних щодо розробки програмного забезпечення довів доцільність пошуку асоціативних правил щодо визначення тривалості виконання процесу розробки програмного забезпечення. Це дозволить автоматизувати процес розробки програмного забезпечення за рахунок визначення тривалості виконання як певного завдання, так і проекту в цілому.

Аналіз доцільності пошуку і використання асоціативних правил щодо тривалості розробки програмного забезпечення показав необхідність використання знайдених залежностей під час процесу розробки програмного забезпечення. Також проведено аналіз методів визначення частих предметних наборів для пошуку таких асоціативних правил. Даний аналіз підтвердив доцільність використання методу визначення частих предметних наборів FP-Growth, який потребує удосконалення з метою виправлення існуючих недоліків та його адаптації до предметної області, що розглядається. Удосконалення полягає у класифікації завдань перед пошуком асоціативних правил, що дозволить підвищити точність визначення тривалості виконання завдання розробником, а також скоротити тривалість пошуку асоціативних правил, які для цього використовуються. Для визначення методу класифікації даних було проведено відповідний аналіз, який показав необхідність застосування методу дерев рішень, оскільки потрібно отримати легко інтерпретовані дані за короткий термін.

Визначено основні задачі дисертаційного дослідження.

Основні результати наукового дослідження опубліковано в [25] – [31].

РОЗДІЛ 2

РОЗРОБКА ТЕОРЕТИКО-МЕТОДИЧНИХ ЗАСАД ПРОЦЕСУ ПОШУКУ АСОЦІАТИВНИХ ПРАВИЛ ЩОДО ТРИВАЛОСТІ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Проведене дослідження методів інтелектуального аналізу даних, а саме пошуку і використання асоціативних правил щодо тривалості розробки програмного забезпечення, показало необхідність їх використання під час даного процесу. Це полегшить роботу менеджерів проектів при плануванні робіт, що здійснюватимуться під час розробки ПЗ.

Для цього необхідно розробити інформаційну модель процесу пошуку асоціативних правил щодо тривалості розробки програмного забезпечення, в основі якої лежить удосконалений метод FP-Growth визначення частих предметних наборів для пошуку таких правил. А також розробити алгоритм використання менеджерами проектів знайдених асоціативних правил для визначення часу, необхідного для виконання завдання розробником.

2.1 Інформаційна модель процесу пошуку асоціативних правил щодо тривалості розробки програмного забезпечення

Враховуючи, що інформаційна модель процесу пошуку асоціативних правил щодо тривалості розробки програмного забезпечення відображає його вхідні та вихідні значення, важливі для даного процесу параметри і величини, а також їхні зв'язки, то її можна подати у вигляді AR :

$$AR = \langle T, D, minsupp, minconf, \mu, \Omega, NT, PT \rangle \quad (2.1)$$

де T – множина завдань, що були виконані при розробці програмного забезпечення;

D – множина розробників, які виконують завдання;

$minsupp$ – мінімальне значення підтримки асоціативних правил;

$minconf$ – мінімальне значення достовірності асоціативних правил;

μ – метод, за допомогою якого буде здійснюватися пошук асоціативних правил;

Ω – множина знайдених асоціативних правил;

NT – множина завдань, для яких необхідно визначити тривалість їх реалізації розробником певного рівня кваліфікації.

PT – множина завдань з визначеним часом, який необхідний на їх виконання розробником, що має певну кваліфікацію.

Нехай множина завдань, що виконані під час розробки програмного забезпечення T відображається виразом:

$$T = \{T_1, T_2, \dots, T_i, \dots, T_n\}, \quad (2.2)$$

де T_i – це i -е виконане завдання, що має характеристики, наведені у виразі 2.3,

$i = \overline{1, n}$, n – потужність множини T .

Нехай i -е завдання T_i представлено виразом:

$$T_i = \langle Tp, Pr, Sv, Cm, Cx, t, Dl, Sx, Rl \rangle, \quad (2.3)$$

де Tp – це тип i -го завдання, що має таку множину значень {покращення, особливість, дефект}.

Pr – це пріоритет (ступінь важливості) i -го завдання, що має таку множину значень {high, medium, low};

Sv – це критичність i -го завдання, що має таку множину значень {blocking, critical, high, medium, low};

Cm – це компонент розроблюваного ПЗ, множина значень якого залежить від конкретного ПЗ;

Cx – це складність завдання, яке потрібно виконати під час розробки ПЗ, що має множину значень {easy, medium, difficult};

t – це час, необхідний для виконання завдання. Значення даної величини залежить від конкретного завдання та розробника, що буде його виконувати;

DI – рівень кваліфікації розробника, що виконав *i*-е завдання, що має таку множину значень {trainee, junior, middle, senior, architect};

Sx – стать розробника, що виконав *i*-е завдання, що має таку множину значень {men, women};

RI – номер реліз, під час якого було виконано *i*-е завдання, множина значень якого залежить від тривалості розробки конкретного ПЗ.

Наведені характеристики є важливими при розгляді будь-якого завдання, оскільки вони враховуються менеджерами проектів при плануванні розробки ПЗ.

Тип – це тип завдання, що виконується під час розробки програмного забезпечення. Залежно від типу завдання, воно виконується на конкретному етапі даного процесу.

Критичність – це рівень впливу даного завдання на розробку або функціонування програмного забезпечення. Даний показник важливий для менеджерів проектів, оскільки він є основним при вирішенні бізнес-питань.

Пріоритет вказує на важливість і порядок, в якому потрібно виконати завдання у порівнянні з іншими.

Компонент – це частина програмного забезпечення, для якої буде виконуватися дане завдання.

Складність показує наскільки дане завдання важке для реалізації розробником. Дана характеристика визначається експертним шляхом, базуючись на досвіді менеджера проекту та розробників.

Час – це тривалість виконання завдання, яку необхідно знати менеджерам проектів при плануванні ресурсів для розробки програмного забезпечення.

Оскільки не всі характеристики завдання однаково впливають на тривалість його розробки, було використано багатофакторний кореляційний аналіз для визначення ступеня зв'язку між характеристиками завдань, що виконуються під час розробки програмного забезпечення, та тривалістю їх розробки.

Оскільки множини можливих значень для характеристик завдань (компонент програмного забезпечення, рівень кваліфікації розробника, тип, складність, пріоритет, критичність завдання, стать розробника та реліз програмного забезпечення) є якісними, то було розраховано коефіцієнт взаємної спряженості Чупрова для вибірки на основі якої здійснювався аналіз (додаток Е).

Результати розрахунку парних коефіцієнтів кореляції характеристик завдань наведено у табл. 2.1, де: ch_1 – тип завдання; ch_2 – рівень кваліфікації розробника; ch_3 – критичність завдання; ch_4 – складність завдання; ch_5 – пріоритет завдання; ch_6 – компонент ПЗ, що розробляється; ch_7 – стать розробника; ch_8 – реліз в якому дане завдання виконується.

Як видно із табл. 2.1 найтісніший зв'язок існує між тривалістю розробки та наступними характеристиками: рівень кваліфікації розробника; тип, критичність, складність, пріоритет завдання та компонент ПЗ, що розробляється. Зв'язок між тривалістю розробки завдання та релізом, під час якого відбувається виконання завдання, нижчий у порівнянні із іншими характеристиками, тому даний фактор не буде враховуватися під час пошуку АП щодо тривалості розробки ПЗ.

Також, як видно з табл. 2.1, найтісніший зв'язок існує між статтю розробника та рівнем кваліфікації – 0.06, що означає, що ці дві характеристики є надлишковими. Якщо порівняти вплив кожної характеристики на тривалість виконання завдання, то можна побачити, що рівень кваліфікації має вищий показник рівня кореляції ніж стать розробника, а тому повинен використовуватися під час пошуку АП щодо тривалості розробки ПЗ.

Таблиця 2.1 – Матриця коефіцієнт кореляції між характеристиками завдань та тривалістю їх виконання

	t	ch_1	ch_2	ch_3	ch_4	ch_5	ch_6	ch_7	ch_8
t	1	0,11	0,15	0,14	0,12	0,14	0,12	0,11	0,05
ch_1	0,11	1	0,02	0,04	0,03	0,04	0,01	0,02	0,01
ch_2	0,15	0,02	1	0,04	0,05	0,04	0,02	0,06	0,03
ch_3	0,14	0,04	0,04	1	0,02	0,03	0,01	0,03	0,02
ch_4	0,12	0,03	0,05	0,02	1	0,02	0,02	0,03	0,01
ch_5	0,14	0,04	0,04	0,03	0,02	1	0,01	0,03	0,01
ch_6	0,12	0,01	0,02	0,01	0,02	0,01	1	0,02	0,02
ch_7	0,07	0,02	0,06	0,03	0,03	0,03	0,02	1	0,01
ch_8	0,05	0,01	0,03	0,02	0,01	0,01	0,02	0,01	1

Отже, множина характеристик завдань, серед яких здійснюється пошук асоціативних правил щодо тривалості розробки програмного забезпечення наступна: рівень кваліфікації розробника; тип, критичність, складність та пріоритет завдання; компонент ПЗ, що розробляється.

Нехай множина розробників D , які виконують завдання, відображається у виразі:

$$D = \{D_1, D_2, \dots, D_j, \dots, D_m\}, \quad (2.4)$$

де D_j – це j -й розробник, $j = \overline{1, m}$,

m – потужність множини D .

Нехай j -й розробник, який виконуватиме завдання D_j представлений виразом:

$$D_j = \langle N, Dl, Sx \rangle, \quad (2.5)$$

де $j = \overline{1, m}$, m – кількість розробників;

N – це ім'я j -го розробника;

Dl – рівень кваліфікації j -го розробника;

Sx – стаття j -го розробника.

Нехай множина транзакцій TS , у яких зустрічаються виконані завдання, визначається як:

$$TS = \{TS_1, TS_2, \dots, TS_k, \dots, TS_K\}, \quad (2.6)$$

де TS_k – це k -а транзакція: i -е завдання, виконане j -м розробником за час t_{ij} , і яку можна подати у вигляді: $TS_k = \{T_i \cap D_j \cap t_{ij}\}$, де $k = \overline{1, K}$, K – потужність множини TS_k .

Множина транзакцій, що складається із завдань T_i , які мають однакові значення характеристик Pr, Sv, t описується виразом:

$$TS_{T_i} = \left\{ \begin{array}{l} T_i \cap D_j \cap t_{ij}, \\ T_i \in T, i = \overline{1, n} \\ D_j \in D, j = \overline{1, m} \end{array} \right\} \subseteq TS \quad (2.7)$$

Підтримка набору $supp(TS_{T_i})$ є відношенням кількості транзакцій, в які входить набір TS_{T_i} , до загальної кількості транзакцій бази транзакцій TS :

$$supp(TS_{T_i}) = \frac{|TS_{T_i}|}{|TS|}. \quad (2.8)$$

Перед пошуком асоціативних правил потрібно визначити *minsupp*, що визначається експертом.

Отже, під час пошуку асоціативних правил щодо тривалості розробки програмного забезпечення необхідно знайти таку множину усіх наборів *LT*, у яких значення підтримки більше за значення *minsupp*, що задане експертом:

$$LT = \{TS_{T_i} | supp(TS_{T_i}) > minsupp\} \quad (2.9)$$

Знайдені асоціативні правила утворюють множину Ω . Нехай *d*-те АП:

$$\Omega_d = \{T_i \cap Dl_j \rightarrow t_{i_j}, T_i \cap Dl_j \cap t_{i_j} \rightarrow \emptyset\} \quad (2.10)$$

де $d = \overline{1, G}$, *G* – кількість асоціативних правил.

t_{i_j} – це тривалість виконання *i*-го завдання розробником з *j*-м рівнем кваліфікації.

Множину *NT*, утворюють завдання, які потрібно виконати під час розробки ПЗ і для яких не визначено тривалість їх виконання. Нехай *f*-е завдання NT_f буде представлено виразом:

$$NT_f = \langle Tp, Pr, Sv, Cm, Cx \rangle, \quad (2.11)$$

де $f = \overline{1, F}$, *F* – кількість завдань.

Множина *PT* складається із завдань для яких визначено тривалість їх виконання. Нехай *g*-е завдання з часом, необхідним на його виконання розробником з певним рівнем кваліфікації PT_g відображено у виразі:

$$PT_g = \langle Tp, Pr, Sv, Cm, Cx, Dl_j, t_{g_j} \rangle, \quad (2.12)$$

де $g = \overline{1, M}$, M – кількість завдань;

t_{g_j} – це тривалість виконання g -го завдання j -м розробником з певним рівнем кваліфікації.

Інформаційну модель процесу пошуку асоціативних правил щодо тривалості розробки програмного забезпечення наведено на рис. 2.1.

Дана інформаційна модель розділена на два окремих блоки:

1. У блоці пошуку асоціативних правил щодо тривалості розробки програмного забезпечення представлено процес пошуку асоціативних правил у БД завдань, що виконувалися розробниками.

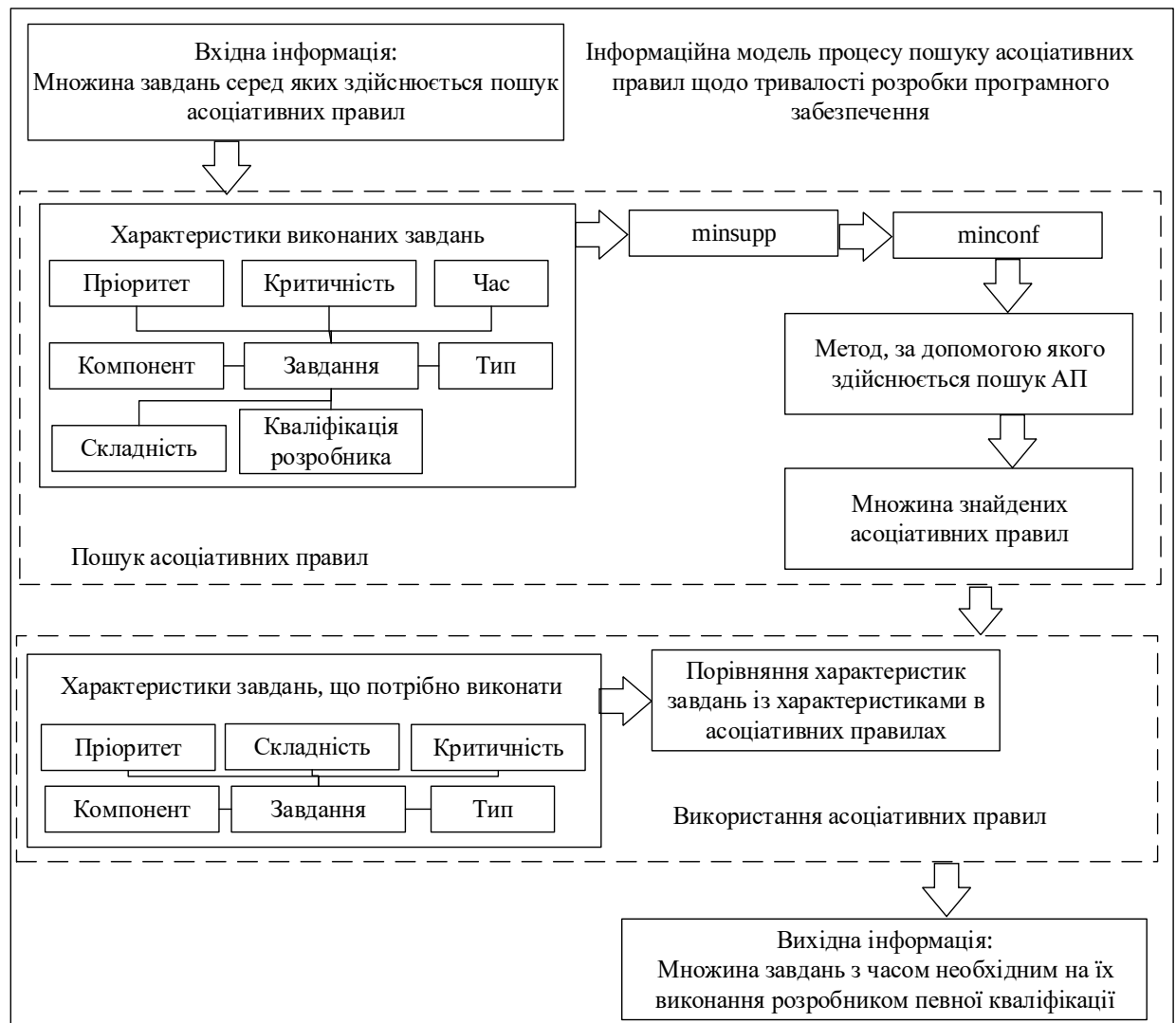


Рисунок 2.1 – Інформаційна модель процесу пошуку асоціативних правил щодо розробки програмного забезпечення

При пошуку асоціативних правил потрібно вказати значення мінімальної підтримки *minsupp* і мінімальної достовірності *minconf*. Зазвичай їх задає експерт в обраній предметній області: під час розробки ПЗ ним являється менеджер проекту.

Результатом виконання даного процесу є множина знайдених асоціативних правил *AR*. Знайдені залежності використовуються для визначення часу, необхідного для виконання завдання конкретним розробником у наступному блоці інформаційної моделі.

2. У блоці використання асоціативних правил представлено процес використання знайдених асоціативних правил щодо тривалості розробки ПЗ. Вони застосовуються для визначення часу, необхідного на виконання завдання конкретним розробником.

Результатом даного процесу є множина завдань PT_g з часом, необхідним на їх реалізацію розробником із відповідним рівнем кваліфікації.

Знайдені асоціативні правила щодо тривалості розробки програмного забезпечення дозволять визначити час, необхідний для виконання завдання розробником певного рівня кваліфікації.

Оскільки розробка ПЗ розглядається з точки зору менеджера проекту, то необхідно вирішити задачу прийняття рішень щодо відбору завдань, які можна реалізувати у визначені терміни та з використанням вказаного бюджету.

Такий план можна представити множиною Y , що представлено у виразі 2.13:

$$\Psi = \langle PT, B, Y, \Delta \rangle, \quad (2.13)$$

де B – бюджет, виділений на розробку ПЗ,

Y – запланована тривалість розробки ПЗ,

Δ – допустиме відхилення тривалості та вартості розробки.

Бюджет – це максимальна сума коштів, що може бути витрачена на розробку даного програмного забезпечення чи його складової. Величина даної характеристики відрізняється залежно від проекту.

Запланована тривалість розробки – це час, який можна витратити на розробку програмного забезпечення чи його складової. Величина даної характеристики також залежить від проекту.

Величина допустимого відхилення тривалості та вартості розробки встановлюється менеджером проекту, залежно від проекту.

Отже, в результаті проведеного аналізу визначено характеристики завдань, які впливають на тривалість виконання завдання і можуть використовуватися під час пошуку асоціативних правил щодо тривалості розробки програмного забезпечення. Такими характеристиками є: рівень кваліфікації розробника; тип, критичність, складність та пріоритет завдання; компонент ПЗ, що розробляється. Розроблено інформаційну модель процесу пошуку асоціативних правил щодо тривалості розробки програмного забезпечення. Дана інформаційна модель, на відміну від існуючих, дозволяє здійснювати пошук асоціативних залежностей серед виконаних під час розробки програмного забезпечення завдань із урахуванням таких характеристик: складність, пріоритет, критичність та тривалість виконання завдання. Знайдені залежності використовуються для визначення часу, необхідного на реалізацію завдання конкретним розробником, що автоматизує процес розробки програмного забезпечення.

2.2 Удосконалений метод FP-Growth визначення частих предметних наборів для пошуку асоціативних правил

У підрозділі 1.4 було проаналізовано методи визначення частих предметних наборів для пошуку асоціативних правил та зроблено висновок про доцільність застосування методу визначення частих предметних наборів для пошуку асоціативних правил FP-Growth щодо тривалості розробки

програмного забезпечення. З метою усунення існуючих недоліків методу FP-Growth, а саме: щоб розмір FP-дерева не перевищував розмір основної пам'яті комп'ютера; щоб були враховані особливості процесу розробки ПЗ та завдання серед яких здійснюється пошук частих предметних наборів, його було удосконалено.

Означені недоліки методу FP-Growth можна усунути за допомогою його удосконалення, що дозволить підвищити точність визначення тривалості виконання завдань під час розробки ПЗ, а також скоротити час пошуку асоціативних правил щодо тривалості розробки програмного забезпечення. Модифікація методу полягає у тому, що відбуватиметься попередній розподіл всієї множини завдань на три групи відповідно до характеристик завдання:

- завдання категорії 1 (завдання, які легко реалізувати),
- завдання категорії 3 (завдання, які важко реалізувати),
- завдання категорії 2 (завдання між легкими і важкими).

Кожна група завдань аналізуватиметься окремо та паралельно, що дозволить скоротити час на добування частих предметних наборів та, відповідно, пошук асоціативних правил. А знайдені асоціативні правила дозволять точніше визначати тривалість виконання завдання розробником певної кваліфікації.

У табл. 2.2 наведено характеристики завдань, що використовуються під час пошуку асоціативних правил щодо тривалості розробки ПЗ, та множини значення, які вони можуть набувати.

Оскільки на початку розробки програмного забезпечення (кількість реалізованих завдань до 50) база даних завдань, що були розроблені не потужна, то доцільно об'єднати певні характеристики завдань та значень, які вони можуть набувати, з метою отримання асоціативних правил щодо тривалості розробки ПЗ. Тобто для кожного завдання, що було виконано під час розробки ПЗ, менеджером проекту буде встановлено відповідне значення характеристик. Потім здійснено пошук АП та їх подальше використання для

визначення тривалості виконання завдань.

Таблиця 2.2 – Характеристики завдань та множина їх можливих значень

Тип (<i>Tr</i>)	покращення, особливість, дефект
Пріоритет (<i>Pr</i>)	low, medium, high
Критичність (<i>Sv</i>)	low, medium, high, critical, blocking
Кваліфікація розробника (<i>Dl</i>)	trainee, junior, middle, senior, architect
Складність (<i>Cx</i>)	easy, medium, difficult
Компонент (<i>Cm</i>)	індивідуально для кожного ПЗ
Тривалість (<i>t</i>)	індивідуально для кожного завдання

Так складність завдань *Cx* об'єднується із критичністю *Sv* у комплексну характеристику – *CS* множина значень якої {low, medium, high}. Значення характеристики *CS* задаються для кожного завдання менеджером проекту, базуючись на його власному досвіді. Для пріоритету завдання *Pr* множина значень зменшується до двох величин: {high, low}. Рівень кваліфікації розробника *Dl* визначається множиною таких значень: {junior, middle, senior}.

Оскільки тип завдання має чітко визначену множину значень, то вона не потребує змін. Значення характеристик, компонент і тривалість визначаються для кожного ПЗ індивідуально, тому їх неможливо об'єднати у певні категорії. Результати об'єднання характеристик та їх значень наведено у табл. 2.3.

Після того як кількість завдань, виконаних кожним розробником, перевищує 50, але менше 100, менеджер проекту отримав досвід роботи з розробниками і може оцінити складність завдання. Тому можна виділити окремо характеристики складність та критичність, але зберегти об'єднання значень цих характеристики. Множина значень характеристики складність завдань *Cx*: {easy, medium, difficult} та критичність *Sv*: {low, medium, high}. Результати роз'єднання характеристик та їх значень наведено у табл. 2.4.

Таблиця 2.3 – Характеристики завдань та множина їх можливих значень після об'єднання (кількість завдань менше 50)

Тип (<i>Tr</i>)	покращення, особливість, дефект
Пріоритет (<i>Pr</i>)	low, high
<i>CS</i> (<i>Sv</i> + <i>Cx</i>)	low, medium, high
Кваліфікація розробника (<i>DI</i>)	junior, middle, senior
Компонент (<i>Cm</i>)	індивідуально для кожного ПЗ
Тривалість (<i>t</i>)	індивідуально для кожного завдання

Таблиця 2.4 – Характеристики завдань та множина їх можливих значень після об'єднання (кількість завдань перевищує 50, але менше 100)

Пріоритет (<i>Pr</i>)	low, high
Складність (<i>Cx</i>)	easy, medium, difficult
Критичність (<i>Sv</i>)	low, medium, high
Кваліфікація розробника (<i>DI</i>)	junior, middle, senior

Після того як кількість завдань, виконаних кожним розробником, перевищує 100, але менше 150, множина можливих значень характеристики пріоритет *Pr* збільшується до {low, medium, high}, критичності *Sv* – {low, medium, high, critical}, а кваліфікація розробника (*DI*) не змінюється. Характеристики та їх значення наведено у табл. 2.5.

Таблиця 2.5 – Характеристики завдань та множина їх можливих зазначень після об'єднання (кількість завдань перевищує 100, але менше 150)

Пріоритет (<i>Pr</i>)	low, medium, high
Складність (<i>Cx</i>)	easy, medium, difficult
Критичність (<i>Sv</i>)	low, medium, high, critical
Кваліфікація розробника (<i>DI</i>)	junior, middle, senior

Після того як кількість завдань, виконаних кожним розробником, перевищує 150, характеристики завдань, що використовуються для пошуку

асоціативних правил щодо тривалості розробки ПЗ та множини їх можливих значень дорівнюють тим, що наведені у табл. 2.2.

З метою підвищення точності визначення тривалості виконання завдання розробником а також скорочення тривалості пошуку асоціативних правил, які для цього використовуються, метод FP-Growth потрібно удосконалити за рахунок класифікації завдань перед початком пошуку залежностей.

Відповідно до кількості завдань, які виконані кожним розробником під час розробки ПЗ, характеристики, відносно яких здійснюється класифікація завдань відрізняються:

- кількість виконаних завдань розробником до 50 – комплексна величина $CS (S_v + C_x)$, критичність завдання та тривалість його виконання;
- кількість виконаних завдань розробником після 50 – пріоритет завдання, його критичність та тривалість його виконання.

UML-діаграму діяльності удосконаленого методу FP-Growth для пошуку асоціативних правил щодо тривалості розробки програмного забезпечення наведено на рис. 2.2.

У формалізованому вигляді удосконалений метод FP-Growth містить такі основні кроки:

1. Задається значення достовірності *minconf*, відносно якого буде визначатися чи знайдене асоціативне правило корисне для експерта чи ні.

Значення *minconf* задається експертом, яким під час розробки ПЗ виступає менеджер проекту.

2. Відбувається сканування БД, а усі завдання, серед яких здійснюватиметься пошук АП, класифікуються на три класи.

Для класифікації використовується метод дерев рішень, а саме алгоритм класифікації С 4.5.

3. Для кожного класу завдань визначається значення *minsupp*, використовуючи формулу 1.5.

Для обрахунку значення $minsupp$, використовуючи формулу 1.5 спочатку потрібно задати значення таких змінних:

- ініціалізувати значення змінної α .
- ініціалізувати значення змінних β, γ .
- обрахувати значення $minsupp$.

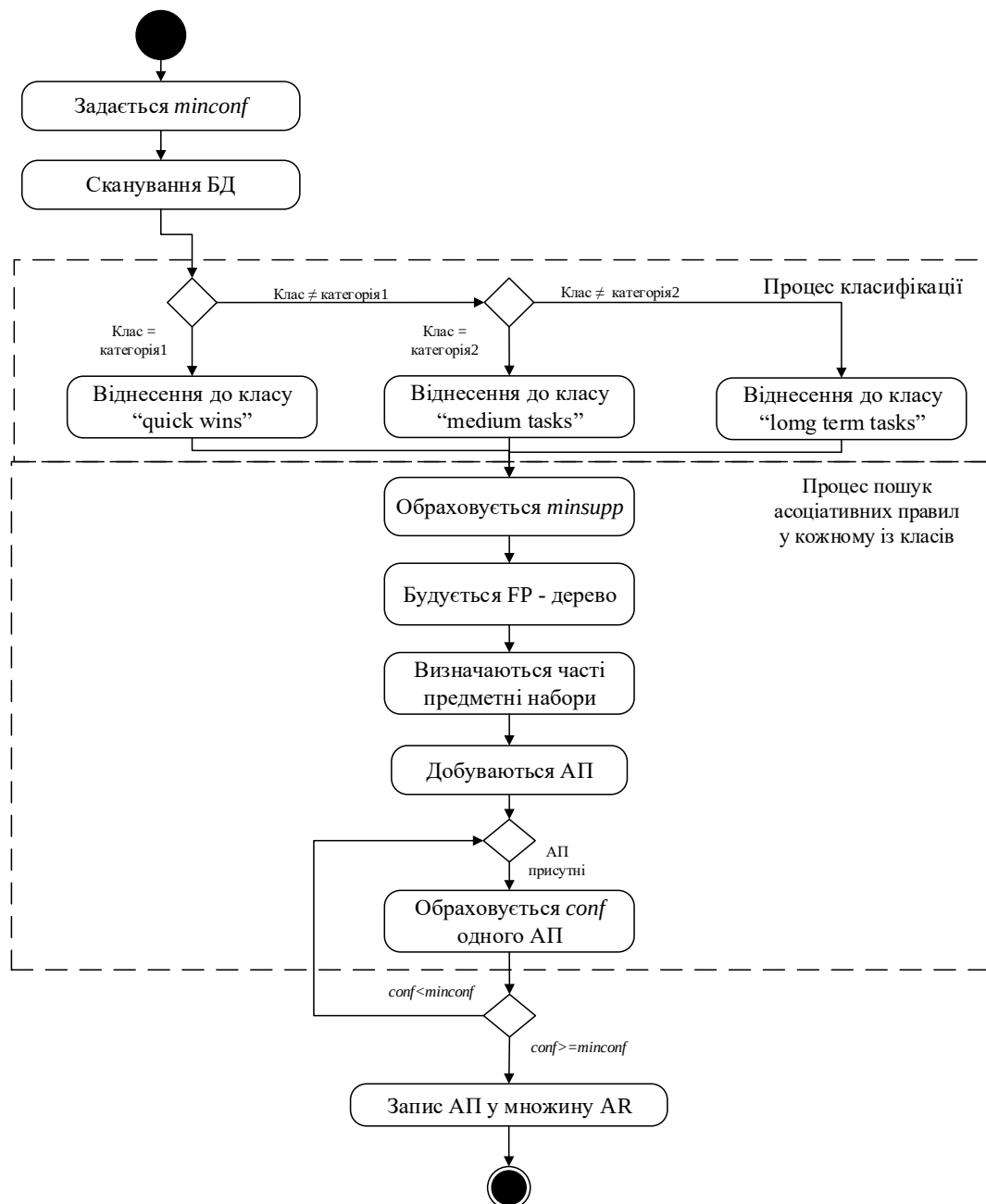


Рисунок 2.2 – UML-діаграма діяльності удосконаленого методу FP-Growth для пошуку асоціативних правил щодо тривалості розробки ПЗ

4. Здійснюється перетворення завдань у кожному класі в деревовидну структуру, що називається FP-дерево.

5. Відбувається визначення частих предметних наборів у FP-дереві з урахуванням обрахованого значення підтримки *supp*.

Оскільки у вхідній базі даних завдання повторюються багаторазово, то в FP-дереві кожне завдання представляється у вигляді вузла, індекс якого вказує на те, скільки разів дане завдання з'являється.

6. Здійснюється пошук асоціативних правил зі знайдених предметних наборів.

7. Обраховується значення достовірності *conf* для кожного асоціативного правила.

8. Відбувається запис усіх асоціативних правил, значення достовірності *conf* яких більша чи рівна значенню *minconf*, до множини асоціативних правил AR.

Оскільки обробка класів відбувається паралельно, то кроки 3-8 даного методу виконуються одночасно для трьох класів завдань.

Отже, для усунення недоліків існуючого методу визначення частих предметних наборів FP-Growth, серед яких здійснюється пошук асоціативних правил щодо тривалості розробки ПЗ, його було удосконалено. Удосконалення полягає у класифікації завдань на три групи залежно від характеристик завдань (складність, пріоритет, критичність та тривалість виконання). Такі характеристики відрізняються залежно від кількості завдань виконаних конкретним розробником під час розробки програмного забезпечення. Дане удосконалення дозволить використовувати метод визначення частих предметних наборів FP-Growth для пошуку асоціативних правил щодо тривалості розробки програмного забезпечення з метою підвищення точності визначення тривалості виконання завдання розробником певної кваліфікації та скорочення часу пошуку таких асоціативних залежностей.

2.3 Алгоритм використання асоціативних правил для визначення часу, необхідного для виконання завдання розробником

Знайдені асоціативні правила щодо тривалості розробки програмного забезпечення можуть бути використані для передбачення часу необхідного на виконання завдання розробником з певним рівнем кваліфікації. Отримана інформація в подальшому може бути використана менеджерами проектів для управління процесом розробки програмного забезпечення та для побудови плану ефективного використання задіяних людських ресурсів.

Асоціативні правила мають вигляд: «якщо завдання T , буде виконуватися розробником з рівнем кваліфікації DI , то тривалість його реалізації дорівнюватиме t ».

У формалізованому вигляді алгоритм використання асоціативних правил щодо тривалості розробки програмного забезпечення складається з таких основних кроків:

1. Відбувається завантаження завдань, для яких необхідно визначити тривалість розробки розробником (множина завдань NT_f).
2. Обирається перше завдання із множини NT_f .
3. Здійснюється відбір асоціативного правила Ω_d із множини асоціативних залежностей, з яким будуть порівнюватися характеристики завдання з пункту 2.
4. Відбувається порівняння значень характеристик завдань з множин NT_f і Ω_d .

Як було зазначено у підрозділі 2.2 на початку розробки ПЗ база даних розроблених завдань є не потужною і тому виникає необхідність в об'єднанні характеристик завдань та їх значень. При використанні знайдених асоціативних правил виникає необхідність у порівнянні характеристики завдань із АП з характеристиками завдань, для яких необхідно визначити тривалість їх розробки.

У табл. 2.6 наведено відповідність об'єднаних характеристик та їх значень, до характеристик та значень, що використовуються для завдань, для яких необхідно визначити тривалість виконання завдання під час розробки ПЗ.

5. Якщо значення характеристик завдань не збігаються, то здійснюється вибір наступного АП з множини знайдених, та здійснюється порівняння значень характеристик наведених у пункті 2.

Таблиця 2.6 – Відповідність об'єднаних характеристик та їх значень

Характеристика	Значення	Відповідність значень характеристик завдань
1	2	3
Кількість розроблених завдань < 50		
Пріоритет (Pr)	low, high	Low = low High = {medium, high}
CS (Sv+ Cx)	low, medium, high	Low = {difficult+low, difficult+medium, easy+low, wasy+medium} Medium = {medium+low, medium+medium, medium+high, difficult+high} High = {easy+ high, easy+ critical, easy+blocking, medium+critical, medium+blocking, difficult+critical, difficult+blocking }
Кваліфікація розробника (DI)	junior, middle, senior	Junior = {trainee, junior} Middle = middle Senior = {senior, architect}
Кількість розроблених завдань > 50 але < 100		
Пріоритет (Pr)	low, high	Low = low High = {medium, high}
Складність (Cx)	easy, medium, difficult	Easy = easy Medium = medium Difficult =difficult

Продовження таблиці 2.6

1	2	3
Критичність (<i>Sv</i>)	low, medium, high	Low = low Medium = medium High = {high, critical, blocking}
Кваліфікація розробника (<i>Dl</i>)	junior, middle, senior	Junior = {trainee, junior} Middle = middle Senior = {senior, architect}
Кількість розроблених завдань > 100 але < 150		
Пріоритет (<i>Pr</i>)	low, medium high	Low = low Medium = medium High = high
Складність (<i>Cx</i>)	easy, medium, difficult	Easy = easy Medium = medium Difficult = difficult
Критичність (<i>Sv</i>)	low, medium, high critical	Low = low Medium = medium High = high, Critical = {critical, blocking}
Кваліфікація розробника (<i>Dl</i>)	junior, middle, senior architect	Junior = {trainee, junior} Middle = middle Senior = senior Architect = architect
Кількість розроблених завдань > 150		
Значення характеристик відповідають наведеним у табл. 2.2		

6. Якщо характеристики збігаються, то для даного завдання визначається тривалість його розробки t розробником з рівнем кваліфікації Dl . Ця інформація отримується з відповідного асоціативного правила.

7. Отримані дані: час t та рівень кваліфікації розробника Dl додаються до завдання, яке записується до множини завдань, для яких визначено час PT . Інформація з неї використовується в подальшому для визначення тривалості розробки ПЗ та побудови оптимального плану розробки ПЗ.

8. У випадку, коли значення характеристик усіх АП не збігаються зі значеннями характеристик нового завдання – воно заноситься до множини завдань, для яких тривалість розробки не визначено *UT*.

9. Для завдань з цієї множини тривалість розробки визначається шляхом опитування розробників. Після визначення часу, необхідного на їх розробку розробником певної кваліфікації, вони також заносяться до множини *PT*.

На основі опису побудовано UML-діаграму діяльності процесу використання асоціативних правил щодо тривалості розробки ПЗ, яку наведено на рис. 2.3.

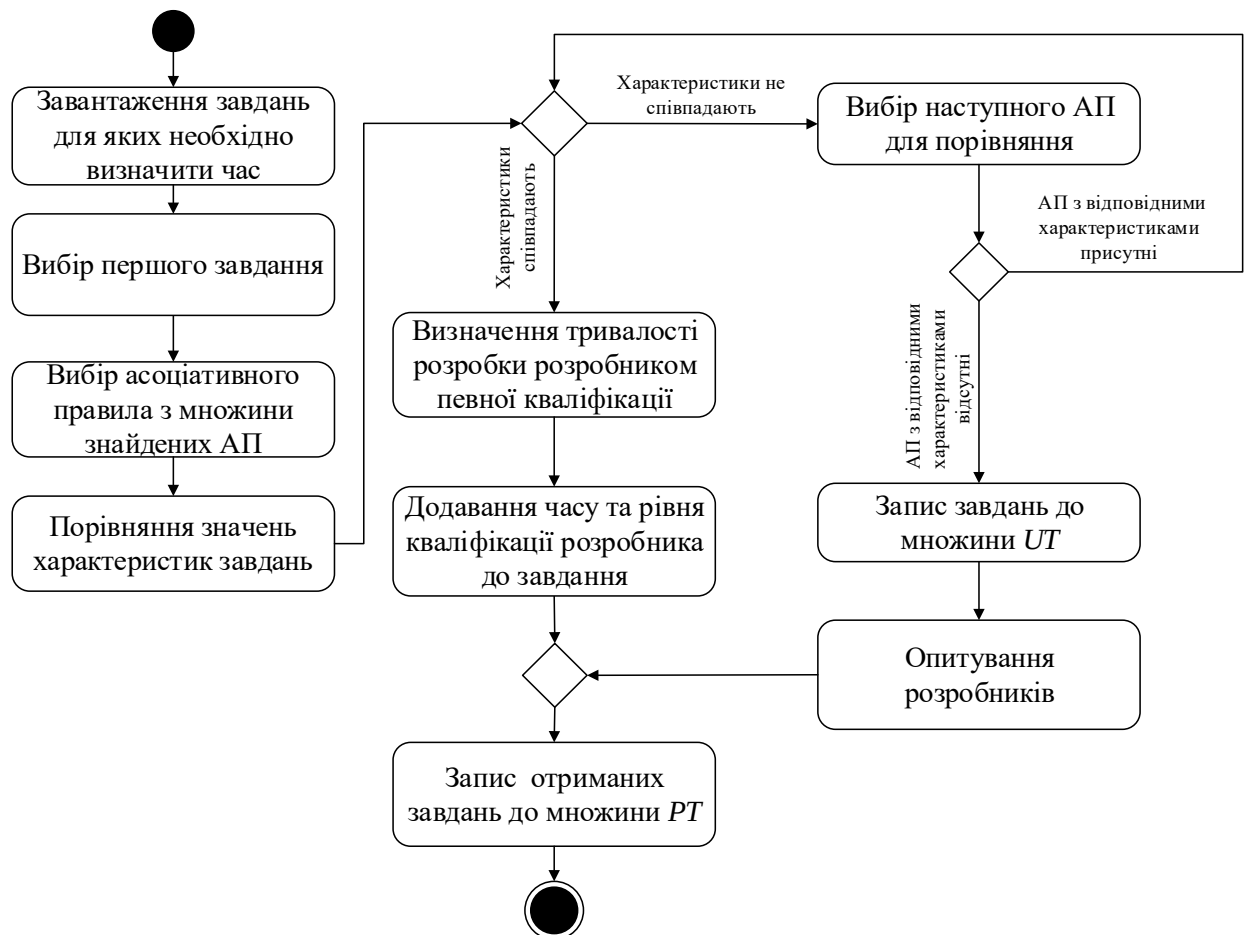


Рисунок 2.3 – UML-діаграма діяльності процесу використання асоціативних правил щодо тривалості розробки програмного забезпечення

Процес розробки ПЗ – динамічний і більшість сучасного програмного забезпечення розробляється з використанням Agile технологій, а саме Scrum. Одним із основних елементів у Scrum є проміжок часу за який команда створює частину програми, що може бути представлена замовнику – «sprint». Під час таких проміжків часу відбувається накопичення нових розроблених завдань у БД, тому виникає необхідність у пошук нових можливих АП після закінчення кожного «sprint». Такий підхід дозволить точніше визначати тривалість виконання завдання розробником певної кваліфікації, оскільки нові АП щодо тривалості розробки ПЗ будуть формуватися на основі більш потужної вибірки даних.

На рис. 2.4 наведено UML-діаграма процесу розробки ПЗ з використанням знайдених АП щодо тривалості розробки ПЗ.

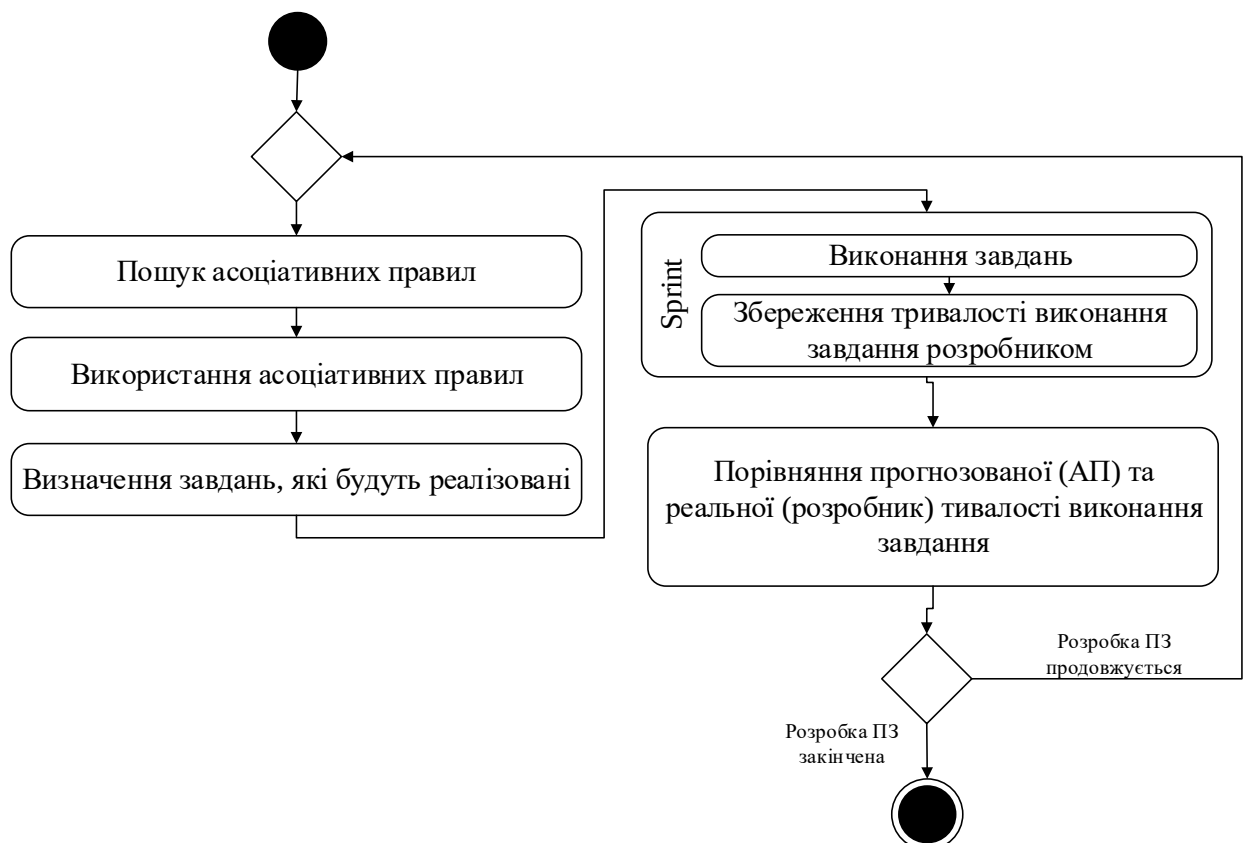


Рисунок 2.4 – UML-діаграма процесу розробки ПЗ з використанням знайдених АП щодо тривалості розробки ПЗ

Перед початком sprint відбувається пошук АП з використанням

удосконаленого методу визначення частих предметних наборів FP-Growth. Знайдені АП використовуються для визначення тривалості виконання завдання розробником певної кваліфікації. Враховуючи характеристики завдань (складність, пріоритет, критичність та тривалість виконання завдання) та визначену тривалість виконання завдань, менеджером проекту відбираються завдання для розробки під час sprint. Під час sprint відбувається реалізація завдання розробником, і фактична тривалість виконання завдання зберігається в БД. Після закінчення sprint відбувається порівняння прогнозованої та реальної тривалості виконання завдання, а також здійснюється пошук нових АП щодо тривалості виконання завдання, що будуть використані для визначення тривалості завдання для наступних sprint.

Отже, розроблено алгоритм використання асоціативних правил щодо тривалості розробки програмного забезпечення та побудовано UML-діаграму діяльності відповідного процесу. В результаті використання даного алгоритму для кожного завдання можна визначати тривалість його реалізації розробником певного рівня кваліфікації. В подальшому така інформація використовуватиметься менеджерами проектів для визначення тривалості розробки ПЗ, що відображено у запропонованому підході до процесу розробки ПЗ з використанням знайдених АП щодо тривалості його розробки.

2.4 Висновки до розділу 2

Результати розрахунку та аналізу коефіцієнтів кореляції характеристик завдань показали, що рівень кваліфікації розробника, тип, критичність, складність та пріоритет завдання, а також компонент ПЗ впливають на тривалість розробки завдання на відміну від таких характеристик як стать розробника та реліз, під час якого виконується завдання. На основі отриманих даних розроблено інформаційну модель процесу пошуку асоціативних правил щодо тривалості розробки програмного забезпечення, що може бути використана менеджерами для визначення тривалості

розробки ПЗ та для планування такого процесу.

З метою усунення недоліків методу визначення частих предметних наборів FP-Growth його було удосконалено за рахунок класифікації завдань перед початком пошуку асоціативних правил на три класи відповідно до характеристик завдань, що відрізняються залежно від кількості завдань виконаних розробником. Таке удосконалення дозволило підвищити точність визначення тривалості виконання завдання розробником певної кваліфікації та скоротити час пошуку асоціативних правил щодо тривалості розробки програмного забезпечення, які для цього використовуються.

Розроблено алгоритм використання знайдених асоціативних правил щодо тривалості розробки програмного забезпечення для визначення часу, необхідного на виконання завдання розробником певного рівня кваліфікації. Даний алгоритм використовується у запропонованому підході щодо розробки програмного забезпечення з використанням знайдених асоціативних правил, що дозволить автоматизувати даний процес.

Основні результати наукового дослідження опубліковано в [32] – [34].

РОЗДІЛ 3

ПОШУК АСОЦІАТИВНИХ ПРАВИЛ ЩОДО ТРИВАЛОСТІ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ З ВИКОРИСТАННЯМ ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ

На основі запропонованої інформаційної моделі пошук асоціативних правил щодо тривалості розробки програмного забезпечення розроблено інформаційну технологію відповідного процесу. Її використання дозволить підвищити точність визначення тривалості виконання завдання розробником певної кваліфікації а також скоротити тривалість пошуку таких правил.

3.1 Інформаційна технологія пошуку асоціативних правил щодо тривалості розробки програмного забезпечення

Інформаційна технологія пошуку асоціативних правил щодо тривалості розробки програмного забезпечення представляє собою набір методів та програмно-технологічних засобів пошуку АП, об'єднаних у технологічний ланцюг. Він забезпечує накопичення, зберігання й аналіз даних про асоціативні правила з метою отримання нової інформації про такі залежності, знайдені щодо розробки програмного забезпечення.

Основні кроки автоматизованої інтелектуальної обробки даних з метою пошуку асоціативних правил такі:

1. Завантаження вхідних даних, необхідних для навчання алгоритму класифікації С 4.5.
2. Процес навчання алгоритму класифікації С 4.5 для подальшого його використання при класифікації завдань, серед яких необхідно знайти асоціативні правила.
3. Завантаження вхідних даних, що стосуються завдань, які було виконано під час розробки ПЗ і серед яких буде здійснюватися пошук асоціативних правил.

4. Процес класифікації вхідної множини завдань (серед яких необхідно знайти асоціативні правила) на три класи, залежно від характеристик завдань.

5. Внесення значення мінімальної достовірності *minconf* експертом, в ролі якого виступає менеджер проекту, та обрахунок значення мінімальної підтримки для кожної групи завдань *minsupp*, використовуючи формулу 2.15.

6. Для кожної групи завдань виконуються пункти 7 – 11.

7. Перетворення завдань у кожному із класів на деревовидну структуру даних FP-дерево.

8. Процес визначення частих предметних наборів у кожному із класів та підрахунок значення підтримки *supp* для кожного із знайдених частих предметних наборів.

9. Відбір частих предметних наборів, у яких значення підтримки *supp* більше або дорівнює значенню мінімальної підтримки *minsupp*, заданого експертом.

10. Процес пошуку асоціативних правил, серед відібраних частих предметних наборів та обрахунок значення достовірності *conf* для кожного із них.

11. Відбір асоціативних правил для яких значення підтримки *conf* більше або дорівнює значенню мінімальної підтримки *minconf*. Це необхідно зробити для визначення асоціативних правил, що є корисними для експерта, в ролі якого виступає менеджер проекту.

12. Збереження відібраних з кожного класу асоціативних правил для їх подальшого використання при визначенні тривалості виконання завдання розробником певної кваліфікації.

13. Процес використання знайдених асоціативних правил щодо тривалості розробки програмного забезпечення з метою визначення часу, необхідного на виконання завдання розробником певної кваліфікації.

З використанням запропонованої інформаційної технології було

розроблено інформаційну систему пошуку асоціативних правил щодо тривалості розробки програмного забезпечення.

Графічне зображення ІТ пошуку асоціативних правил щодо тривалості розробки програмного забезпечення наведено на рис. 3.1.



Рисунок 3.1 – Інформаційна технологія пошуку асоціативних правил щодо тривалості розробки програмного забезпечення

На діаграмі компонентів інформаційної технології пошуку асоціативних правил щодо тривалості розробки програмного забезпечення наведено відповідні компоненти та їх взаємозв'язок. Так робота модуля класифікації завдань залежить від результатів, отриманих модулем навчання алгоритму класифікації, який використовує завдання для навчання відповідного алгоритму. Модуль класифікації завдань використовує завдання серед яких здійснюватиметься пошук АП, і передає результати класифікації у модуль обрахунку значення *minsupp*. Потім у модулі визначення частих предметних наборів, використовуючи обраховані значення мінімальної підтримки, здійснюється визначення частих предметних наборів. Отримані дані передаються в модуль пошуку асоціативних правил, де здійснюється пошук АП, а потім у модулі використання асоціативних правил відбувається процес їх обробки та прийняття рішень щодо використання.

Діаграма компонентів інформаційної технології пошуку асоціативних правил щодо тривалості розробки програмного забезпечення відображає залежності між складовими, якими виступають модулі наведені на рис. 3.2.

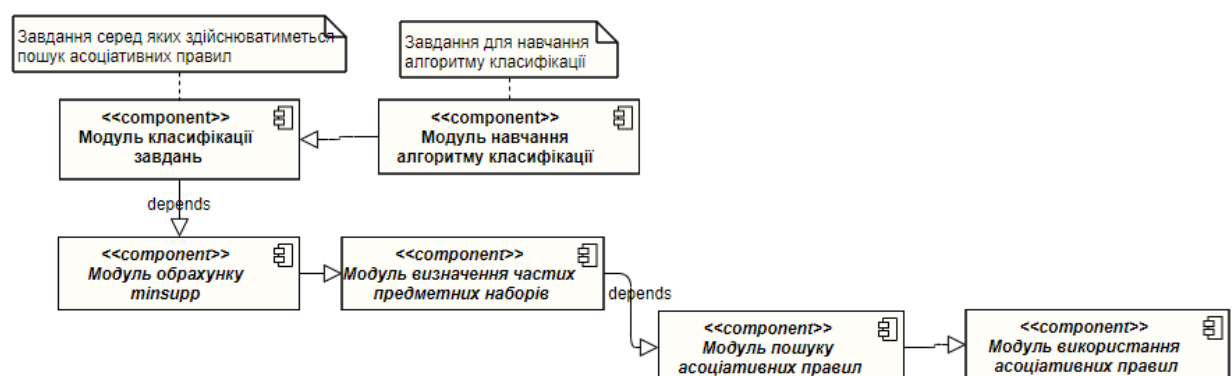


Рисунок 3.2 – UML – діаграма компонентів інформаційної технології пошуку АП щодо тривалості розробки ПЗ

Як видно з рис. 3.2, ІТ пошуку асоціативних правил щодо тривалості розробки програмного забезпечення складається з шести компонентів, які взаємопов'язані між собою.

Отже, розроблено інформаційну технологію пошуку асоціативних правил щодо тривалості розробки програмного забезпечення. Запропонована ІТ базується на розробленій інформаційній моделі, що дозволило скоротити час, необхідний для пошуку асоціативних правил, а також збільшити точність визначення часу, необхідного на реалізацію завдання розробником певної кваліфікації. Наведено UML – діаграму компонентів запропонованої ІТ пошуку асоціативних правил щодо тривалості розробки ПЗ.

3.2 Моделювання процесу розробки програмного забезпечення

Розробку програмного забезпечення можна розглядати на трьох рівнях деталізації кожен з яких можна відобразити за допомогою відповідного математичного апарату. Такі рівні наведено на рис. 3.3:

- I рівень, що представляє розробку ПЗ як скінченну множину компонентів;
- II рівень, що представляє детальний опис етапів розробки конкретного компонента;
- III рівень, що представляє детальний опис окремого етапу розробки конкретного компонента.

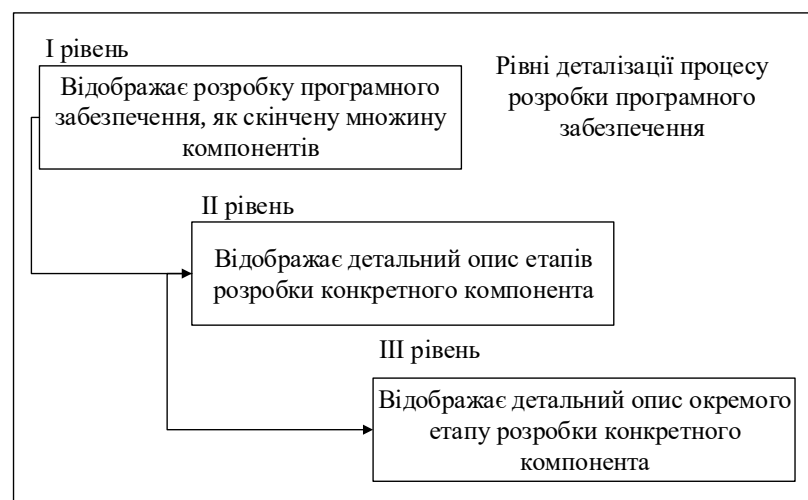


Рисунок 3.3 – Рівні деталізації розробки компонентів програмного забезпечення

Процес розробки ПЗ, що складаються із багатьох компонентів (більше 3) є процесом, під час якого змінюється стан програмного забезпечення та його компонентів у часі. Розробка програмного забезпечення та його компонентів залежить від параметрів програмного забезпечення і не залежить від засобів та способів визначення цих параметрів. Оскільки у даній роботі розглядається спрощений процес розробки програмного забезпечення, то стан такого програмного забезпечення чи його компонентів залежить від попереднього стану і не залежить від історії перебування у попередніх станах, тому такий процес може бути розглянутий з використанням марковських ланцюгів з дискретними станами та дискретним часом [143], [144].

Оскільки модель – це описове представлення об'єкта чи явища, що є спрощеною версією модельованого об'єкта, проекту чи явища [145], то на першому рівні деталізації програмне забезпечення *Product* можна відобразити як скінченну множину окремих компонентів *Product_z* (3.1), що і наведено на рис. 3.4. Такі компоненти можуть використовуватися самостійно або в складі програмного забезпечення.

$$Product = \{Product_1, Product_2, \dots, Product_z, \dots, Product_z\}, \quad (3.1)$$

де $z = \overline{1, Z}$, Z – загальна кількість компонентів у складному програмному забезпеченні.

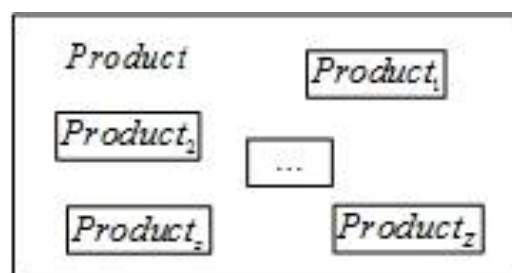


Рисунок 3.4 – Графічне зображення першого рівня деталізації процесу розробки програмного забезпечення

Можливі стани $\dot{S}$, в яких може перебувати та в які може переходити програмне забезпечення на даному рівні, відображені на рис. 3.5:

- $\dot{S}_1$ – ПЗ не розробляється;
- $\dot{S}_2$ – ПЗ знаходиться в процесі розробки;
- $\dot{S}_3$ – ПЗ впроваджується.

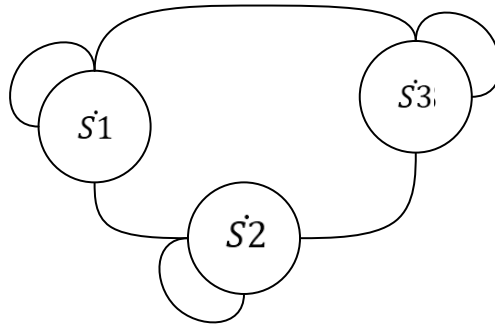


Рисунок 3.5 – Граф станів процесу розробки ПЗ на першому рівні деталізації

Перехід із поточного стану $\dot{S}_l$ в наступний стан $\dot{S}_p$, в момент часу Tt характеризується ймовірністю переходу [146].

Матрицю ймовірностей переходів між станами $\dot{P}$ для процесу розробки ПЗ на першому рівні деталізації наведено у виразі 3.2:

$$\dot{P} = \begin{pmatrix} \dot{p}_{11} & 1 - \dot{p}_{11} & 0 \\ 0 & \dot{p}_{22} & 1 - \dot{p}_{22} \\ 1 - \dot{p}_{33} & 0 & \dot{p}_{33} \end{pmatrix} \quad (3.2)$$

Другий рівень деталізації розробки ПЗ відображає детальний опис етапів розробки конкретного компонента, що і відображено на рис. 3.6.

Кожен з етапів розробки одного компонента програмного забезпечення є станом $\ddot{S}_p$, у який переходить даний компонент у певний момент часу. Тобто другий рівень деталізації можна представити у вигляді графа, наведеного на рис. 3.7, де:

- $\ddot{S}_1$ – аналіз вимог до компонента;
- $\ddot{S}_2$ – опис нового функціоналу та його проектування;
- $\ddot{S}_3$ – програмування та тестування компонента;
- $\ddot{S}_4$ – документування компонента.



Рисунок 3.6 – Графічне зображення етапів другого рівня деталізації процесу розробки програмного забезпечення

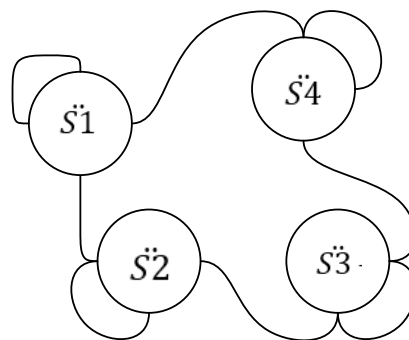


Рисунок 3.7 – Граф станів одного компонента ПЗ на другому рівні деталізації процесу розробки ПЗ

У матриці $\ddot{P}$, що відображена у виразі, наведено ймовірності переходів між станами компонентів програмного забезпечення, що розробляється:

$$\ddot{P} = \begin{pmatrix} \ddot{p}_{11} & 1 - \ddot{p}_{11} & 0 & 0 \\ 0 & \ddot{p}_{22} & 1 - \ddot{p}_{22} & 0 \\ 0 & 0 & \ddot{p}_{33} & 1 - \ddot{p}_{33} \\ 1 - \ddot{p}_{44} & 0 & 0 & \ddot{p}_{44} \end{pmatrix} \quad (3.3)$$

Третій рівень деталізації розробки програмного забезпечення відображає детальний опис окремого етапу розробки конкретного компоненту. Одним із таких етапів є програмування і тестування, основні кроки якого наведено на рис. 3.8.

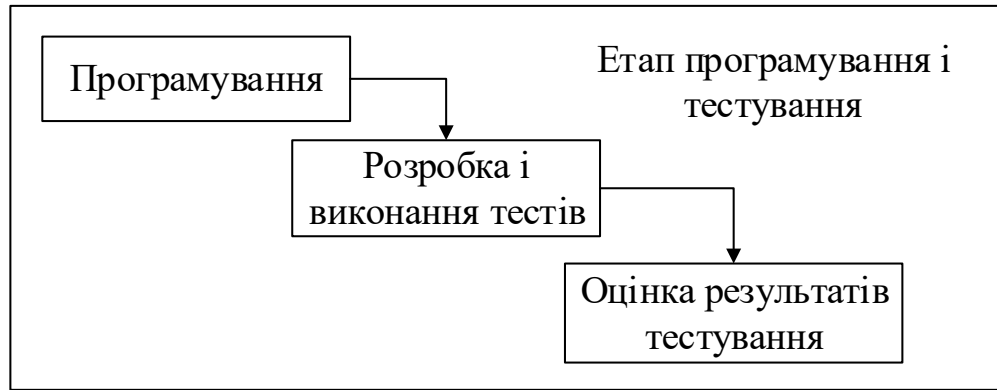


Рисунок 3.8 – Графічне зображення етапу програмування і тестування третього рівня деталізації процесу розробки програмного забезпечення

Аналогічно до попередніх рівнів деталізації, кожен з етапів розробки одного компонента програмного забезпечення можна розглядати як стан $\ddot{S}$, де перехід із поточного стану $\ddot{S}_i$ в наступний стан $\ddot{S}_p$ характеризується ймовірністю переходу, що відображено у виразі 3.4 матриця $\ddot{P}$.

$$\ddot{P} = \begin{pmatrix} \ddot{p}_{11} & 1 - \ddot{p}_{11} & 0 \\ 0 & \ddot{p}_{22} & 1 - \ddot{p}_{22} \\ 1 - \ddot{p}_{33} & 0 & \ddot{p}_{33} \end{pmatrix} \quad (3.4)$$

Третій рівень деталізації можна представити у вигляді графа, який наведено на рис. 3.9, де стани:

- $\ddot{S}_1$ – програмування;
- $\ddot{S}_2$ – розробка і виконання тестів;
- $\ddot{S}_3$ – оцінка результатів тестування.

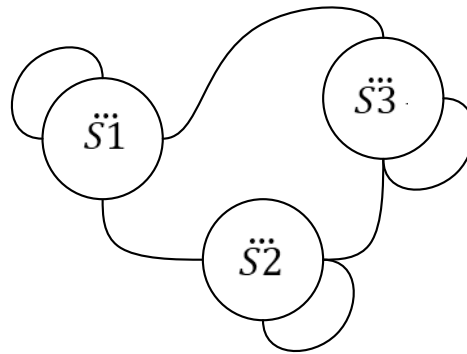


Рисунок 3.9 – Приклад станів графа одного етапу ПЗ на третьому рівні деталізації процесу розробки ПЗ

Отже, для відображення багаторівневого складного процесу розробки програмного забезпечення можна використовувати єдиний математичний апарат, а саме марковський ланцюг.

Представлення процесу розробки програмного забезпечення *PCounter* з використанням марковських ланцюгів було здійснено за таким алгоритмом:

1. Визначено основні компоненти з яких складається дане програмне забезпечення, а саме: Reports, Counter, Routs, Admin, Settings, GPS, Archive, Users.

2. Для кожного компонента визначено основні етапи його розробки, що відображені на рис. 3.7.

3. Для кожного етапу визначено основні кроки з яких він складається.

На кожному із кроків алгоритму відбувався підрахунок компонентів, етапів та кроків відповідно. В результаті проведених досліджень виявлено такі закономірності:

1. При аналізі процесу розробки програмного забезпечення на першому рівні деталізації було виявлено 8 основних компонентів із яких складається комплексне програмне забезпечення *PCounter*. Кількість етапів та кроків процесу розробки вказаного програмного забезпечення на даному рівні не визначена.

2. Аналіз кожного окремого із 8 виявлених компонентів на другому рівні деталізації виявив 4 основних необхідних етапи (рис. 3.6) для його

реалізації. Кількість кроків процесу розробки вказаного програмного забезпечення на даному рівні не визначена.

3. При аналізі кожного етапу на третьому рівні деталізації було встановлено, що етап аналізу, проектування та документування виконується за 1 крок. Етап програмування та тестування передбачає такі основні 3 кроки: програмування, розробка і виконання тестів, оцінка результатів тестування (рис. 3.8). Таким чином, процес розробки одного компоненту програмного продукту *PCounter* на третьому рівні деталізації визначається в 6 кроків.

В табл. 3.1 наведено кількісні показники результату відображення процесу розробки програмного забезпечення для кожного рівня деталізації.

Щодо розробки програмного продукту *PCounter*, необхідно розглянути ($8 \times 4 \times 6 = 192$) основних кроків його реалізації.

Таблиця 3.1 – Результати відображення процесу розробки програмного забезпечення

Рівні деталізації	Кількість компонентів	Кількість етапів	Кількість кроків
I рівень деталізації	8 компонентів	Не визначено	Не визначено
II рівень деталізації	8 компонентів	4 етапи	Не визначено
III рівень деталізації	8 компонентів	4 етапи	6 кроків

Отже, розробку програмного забезпечення можна відобразити на різних рівнях деталізації, кожен з яких можна представити, використовуючи марковський ланцюг.

Використовуючи експертні дані щодо тривалості розробки програмного забезпечення, надані менеджерами проєктів, було розраховано значення ймовірностей переходів між станами для трьох рівнів деталізації процесу розробки програмного забезпечення (додаток В).

Моделювання процесу розробки ПЗ відбувається з використанням вище зазначених даних та складається з наступних етапів. Першим етапом

моделювання процесу розробки ПЗ є завантаження модуля, що здійснює розбиття комплексного програмного забезпечення на основні компоненти. Другим етапом є визначання тривалості періоду, протягом якого змінюється стан кожного окремого компонента програмного забезпечення. Третім етапом є встановлення основних етапів розробки програмного забезпечення, що задаються користувачем; а під час наступного етапу, четвертого, визначається одиниця часу, під час якої відбувається перехід від одного етапу розробки ПЗ до іншого. На п'ятому етапі моделювання користувачем задаються кроки, що здійснюються під час певного етапу розробки ПЗ, оскільки вони відрізняються в залежності від етапу. На шостому етапі моделювання відбувається визначення тривалості переходів між даними кроками розробки ПЗ.

Моделювання процесу розробки ПЗ на I рівні деталізації показало, що зміна стану відбувається кожної умовної одиниці часу, що і відображено на рис. 3.10. Оскільки такий проміжок часу на першому рівні деталізації є великим відносно до загальної тривалості процесу розробки ПЗ, то пошук асоціативних правил щодо тривалості розробки ПЗ недоцільно здійснювати на даному рівні деталізації.

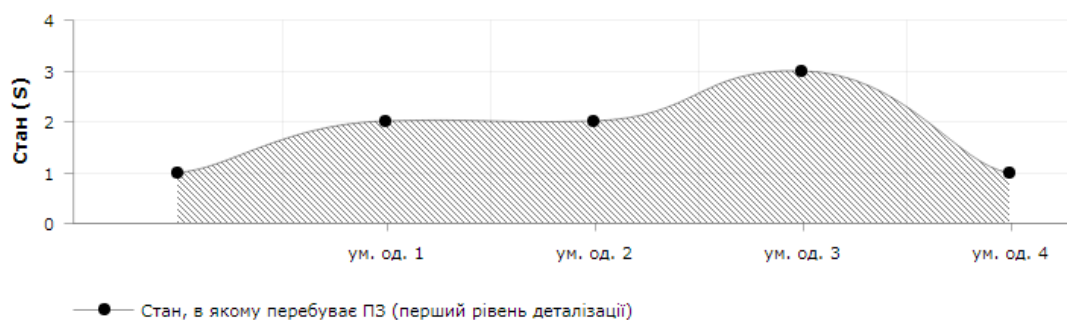


Рисунок 3.10 – Результати моделювання процесу розробки ПЗ на першому рівні деталізації

Моделювання процесу розробки ПЗ на II рівні деталізації показало, що зміна стану відбувається кожної умовної одиниці часу, що і відображено на

рис. 3.11. Оскільки така умовна одиниця є меншою ніж на першому рівні деталізації та є важливою одиницею виміру тривалості розробки ПЗ, то на даному рівні деталізації варто здійснювати пошук асоціативних правил щодо тривалості розробки ПЗ.



Рисунок 3.11 – Результати моделювання процесу розробки ПЗ на другому рівні деталізації

Моделювання процесу розробки ПЗ на III рівні деталізації показало, що зміна стану відбувається кожної умовної одиниці часу, що і відображено на рис. 3.12. А оскільки виникає необхідність у плануванні робочого дня учасників процесу розробки ПЗ, то на даному рівні деталізації варто здійснювати пошук асоціативних правил щодо тривалості розробки ПЗ.

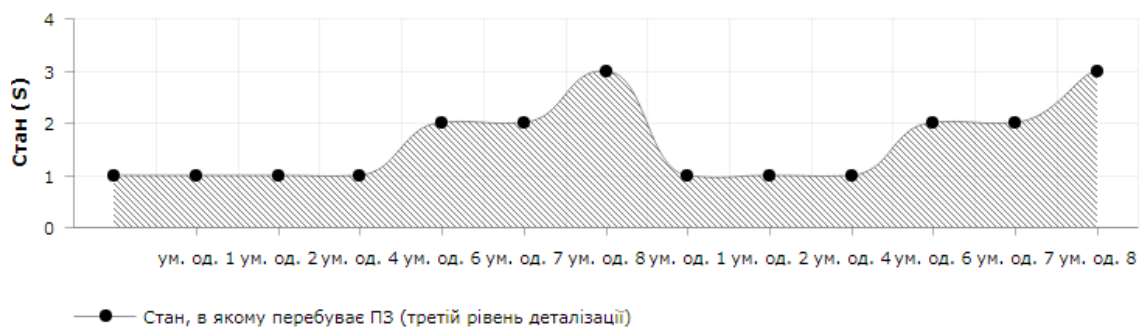


Рисунок 3.12 – Результати моделювання процесу розробки ПЗ на третьому рівні деталізації

Отже, результати представлення процесу розробки ПЗ показали, що зі збільшенням рівня деталізації виникає необхідність пошуку асоціативних

правил щодо тривалості процесу розробки ПЗ. Асоціативні залежності між розробником, що виконує завдання, та часом, необхідним для вирішення поставлених завдань, доцільно шукати лише на рівнях, де процес розробки ПЗ розглядається для окремого компонента а не системи в цілому, тому їх доцільно шукати на другому та третьому рівнях деталізації.

3.3 Моделювання процесу пошуку асоціативних правил щодо тривалості розробки програмного забезпечення з використанням удосконаленого методу FP-Growth

Пошук асоціативних правил з використанням удосконаленого методу визначення частих предметних наборів FP-Growth, на відміну від існуючого методу, передбачає етап віднесення завдань до класів залежно від значень характеристик завдань. Таке удосконалення дозволить підвищити точність визначення тривалості виконання завдання розробником, а також скоротити тривалість пошуку асоціативних правил, які для цього використовуються.

UML-діаграма діяльності процесу пошуку асоціативних правил за допомогою удосконаленого методу FP-Growth відображена на рис. 3.13.

Першим етапом процесу пошуку асоціативних правил з використанням удосконаленого методу FP-Growth є класифікація вхідних даних на три класи.

Даний етап і є відмінністю удосконаленого методу FP-Growth від класичного та складається із двох основних кроків:

1. Спочатку відбувається навчання алгоритму класифікації даних C4.5, який використовується для віднесення завдань до визначених класів.
2. Потім відбувається класифікація завдань, серед яких буде здійснюватися пошук асоціативних правил.



Рисунок 3.13 – UML-діаграма діяльності процесу пошуку асоціативних правил за допомогою удосконаленого методу FP-Growth

Наступним етапом є пошук асоціативних правил у кожному із визначених класів. Він складається із наступних кроків:

1. Обрахунок значення мінімальної підтримки *minsupp* на основі кількості завдань у кожному із класів. Для цього спочатку відбувається внесення експертом значення мінімальної достовірності майбутніх

асоціативних правил *minconf*, на основі якого розраховується значення *minsupp* за допомогою формули 1.5.

2. Відбувається перетворення завдань у кожному із класів у деревовидну структуру FP-дерево та обрахунок значення підтримки для кожного елемента даного дерева.

3. Визначення частих предметних наборів (значення підтримки яких більше чи дорівнює значенню мінімальної підтримки), серед яких здійснюватиметься пошук асоціативних правил щодо тривалості розробки ПЗ.

4. Відбувається обрахування значення достовірності для знайдених асоціативних правил. Цей крок є важливим у даному моделюванні, оскільки без нього неможливо здійснити пошук АП та визначити які із них є корисними для експерта.

5. Серед знайдених АП обираються ті, в яких значення достовірності більше або дорівнює заданому експертом значенню мінімальної достовірності.

Останнім етапом є збереження отриманих асоціативних правил в базі даних, обробка інформації з якої дозволить менеджерам проекту ефективно керувати процесом розробки ПЗ.

Моделювання процесу пошуку асоціативних правил з використанням удосконаленого методу визначення частих предметних наборів FP-Growth проводиться відповідно до математичного опису та методу, який наведено у підрозділі 2.2 дисертаційного дослідження з використанням пакету прикладних програм MATLAB.

У табл. 3.2 наведено результати експериментальних досліджень з використанням відомого методу пошуку АП FP-Growth та удосконаленого методу. Тестова інформація наступна:

- кількість завдань в БД – до 100 елементів, до 250 елементів, до 500 елементів, до 750 елементів, до 1000 елементів;
- час, витрачений на їх пошук.

Таблиця 3.2 – Результати моделювання процесу пошуку асоціативних правил з використанням існуючого методу FP-Growth та удосконаленого

Потужність вибірки (елементів)	Метод	
	FP-Growth	Удосконалений FP-Growth
	Тривалість пошуку (сек)	
100	1.988	1.981
250	3.058	3.021
500	6.258	6.184
750	9.654	9.526
1000	12.754	12.502

На основі отриманих результатів було розраховано, що швидкість пошуку АП з використанням удосконаленого методу FP-Growth при кількості елементів:

- 1) до 100 – зросла на 0,001 сек (0,35%);
- 2) до 250 – зросла на 0,037 сек (1,2%);
- 3) до 500 – зросла на 0,074 сек (1,2%);
- 4) до 750 – зросла на 0,128 сек (1,3%);
- 5) до 1000 – зросла на 0,252 сек (2%).

Кількість знайдених асоціативних правил з використанням удосконаленого методу визначення частих предметних наборів для пошуку АП щодо тривалості розробки ПЗ FP-Growth зменшується зі збільшенням потужності тестової вибірки.

Як видно з табл. 3.2:

- час, необхідний для пошуку асоціативних правил, збільшується зі зростанням кількості завдань у базі даних;
- різниця між часом, що витрачається на пошук асоціативних правил без застосування удосконаленого методу та з його використанням, збільшується зі збільшенням загальної кількості тестових даних.

Отже, експериментальні дослідження показали, що використання ІТ пошуку АП щодо тривалості розробки ПЗ є доцільним у випадку коли

загальна кількість завдань, серед яких потрібно шукати асоціативні правила, понад 500 елементів.

Також було проведено дослідження щодо залежності між кількістю реалізованих завдань під час розробки ПЗ та кількістю знайдених серед них АП щодо тривалості розробки ПЗ та точністю визначення тривалості виконання завдань з використанням таких правил.

Як видно з рис. 3.14 зі збільшенням кількості реалізованих завдань, що зберігаються в БД, збільшується кількість знайдених асоціативних правил. Так серед 50 завдань було знайдено 14 АП, а серед 1000 завдань – 153 правила, а це означає що при збільшенні кількості завдань в 20 разів кількість знайдених АП зросла в 10,9 разів відповідно. Пошук асоціативних правил серед 1500 та 2000 збережених завдань показав, що кількість знайдених АП збільшується (158 та 161 знайдених правила відповідно). Але, оскільки, різниця у 8 знайдених правил при збільшенні кількості завдань на 1000 одиниць не є суттєвою, то можна зробити висновок про недоцільність пошуку АП при кількості виконаних завдань більше 1000.

В свою чергу зі збільшенням кількості асоціативних правил, збільшується точність визначення тривалості виконання завдань, що характеризується похибкою. Результати моделювання відображено на рис. 3.15.

При знайдених 14 асоціативних правилах похибка визначення тривалості виконання завдання розробником становила 12%, а при збільшенні кількості правил до 153, точність зросла в 2,3 рази (значення похибки – 5,2%), що дозволило менеджерам здійснювати ефективне планування процесу розробки програмного забезпечення. Оскільки пошук асоціативних правил не доцільно здійснювати при кількості завдань більше 1000, то мінімальне значення похибки, яку можна досягнути для визначення точності визначення тривалості виконання завдань, – 5,2%.

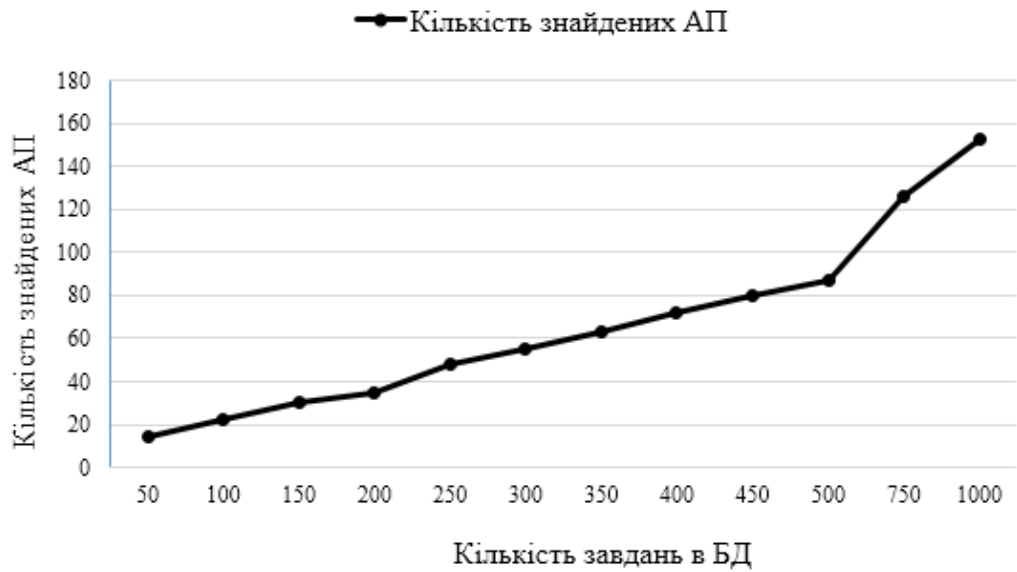


Рисунок 3.14 – Графік зміни кількості знайдених асоціативних правил щодо тривалості процесу розробки ПЗ від кількості завдань у базі даних

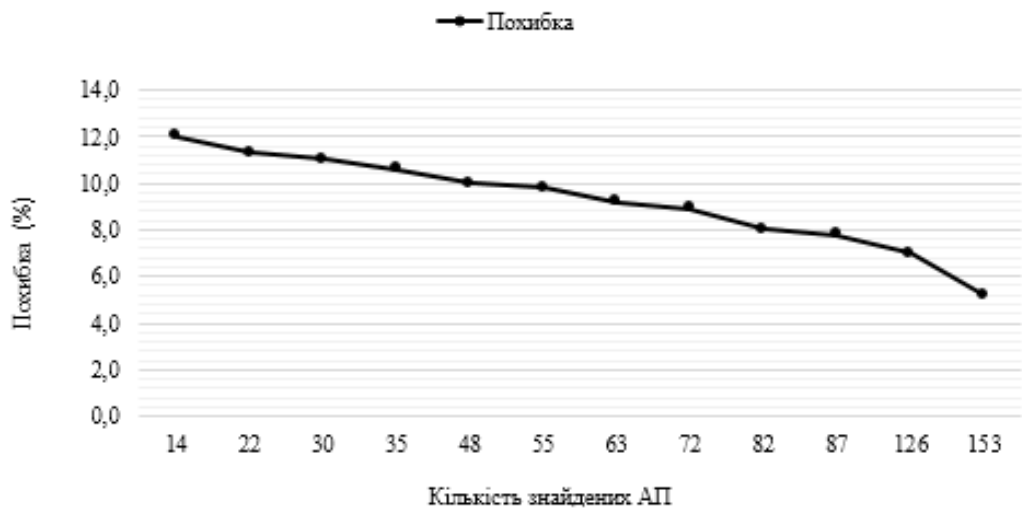


Рисунок 3.15 – Графік зміни похибки визначення тривалості виконання завдань від кількості знайдених асоціативних правил щодо тривалості процесу розробки ПЗ

Оскільки знайдені асоціативні правила використовуються для визначення тривалості виконання завдання розробником певної кваліфікації, то також було промодельовано процес використання таких АП, а результати моделювання наведено у табл. 3.3.

Таблиця 3.3 – Результати використання знайдених асоціативних правил щодо тривалості процесу розробки ПЗ

Кількість завдань	Метод	
	Стандартний підхід (10хв на 1 завдання)	Запропонований підхід (з використанням АП)
	Тривалість визначення часу (хв)	
5	50	10
8	80	16
11	110	22
14	140	28
17	170	42
20	200	40

Зазвичай для визначення тривалості виконання завдання певним розробником відбуваються наради, під час яких кожний із розробників дає свою оцінку часу, необхідного на виконання завдання. Оскільки розробники мають різний рівень кваліфікації, то і оцінка тривалості залежить від досвіду програміста. Відбувається обговорення поставленого завдання і в результаті визначається час, необхідний на його реалізацію. В середньому на визначення тривалості виконання 1 завдання необхідно 10 хв.

Для кожного нового завдання, яке необхідно виконати, виникає необхідність у проведенні таких нарад, що відволікає від трудового процесу, а також вносить зміни в попереднє планування, зроблене менеджером проекту.

Як видно із графіка, наведеного на рис. 3.16, тривалість визначення часу необхідного на виконання завдання, зростає зі збільшенням кількості завдань. Для випадку 17 завдань тривалість виконання одного завдання не була розрахована з використанням знайдених асоціативних правил (і визначалася опитуванням розробників), тому час більший ніж для 20 завдань (для визначення тривалості яких використовувалися лише АП).

Розроблений підхід щодо визначення тривалості виконання завдання

дозволяє використовувати знайдені асоціативні правила для автоматизації процесу визначення часу, необхідного на виконання завдання. Це дозволить скоротити тривалість нарад, не відволікати розробників від трудового процесу, а також не вносити зміни в попередній план процесу розробки програмного забезпечення.

Як видно із рис. 3.16 тривалість визначення необхідного часу на виконання завдання розробником певної кваліфікації скоротилася в 5 разів (із 200 хв при стандартному підході до 40 хв – при запропонованому) для випадку, коли кількість завдань, для яких необхідно визначити час, дорівнює 20. Це свідчить про доцільність використання удосконаленого методу пошуку асоціативних правил щодо тривалості процесу розробки ПЗ, з подальшим використанням знайдених залежностей для визначення тривалості виконання завдання розробником певної кваліфікації.

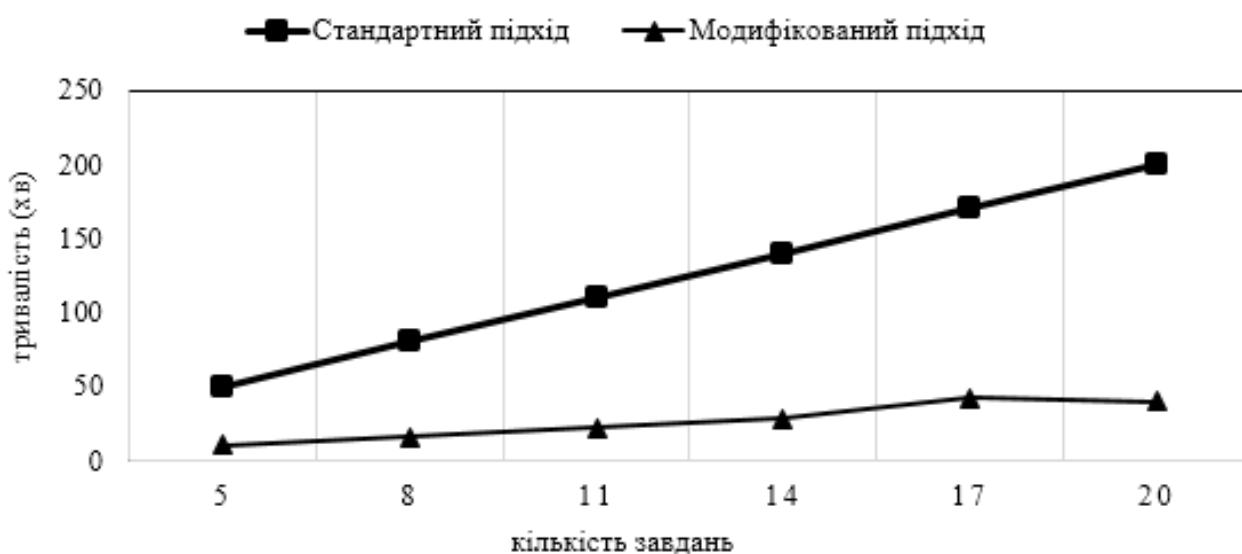


Рисунок 3.16 – Графік зміни тривалості визначення часу, необхідного на виконання завдань, із використанням знайдених асоціативних правил щодо тривалості процесу розробки ПЗ

Отже, проведено моделювання процесу пошуку асоціативних правил щодо тривалості розробки програмного забезпечення з використанням удосконаленого методу FP-Growth. Особливістю даного методу є віднесення

завдань до класів саме перед здійсненням пошуку асоціативних правил. Результати моделювання підтвердили скорочення тривалості пошуку асоціативних правил до 2% за умови, що кількість завдань перевищує 750 елементів.

Проведене моделювання також показало лінійну залежність між кількістю знайдених асоціативних правил щодо тривалості розробки програмного забезпечення та кількістю розроблених завдань у базі даних (кількість знайдених АП зростає в 10,9 разів, у випадку збільшення кількості завдань у базі даних в 20 разів). В результаті моделювання також було отримано залежність точності визначення тривалості виконання завдань від кількості знайдених асоціативних правил. Точність зросла у 2,3 рази при збільшенні кількості асоціативних правил у 11 разів.

Також проведено моделювання процесу використання асоціативних правил з метою визначення тривалості виконання завдання розробником певної кваліфікації. Результати такого моделювання показали зростання швидкості визначення часу, необхідного на виконання завдання розробником до 5 разів, що свідчить про доцільність використання асоціативних правил щодо тривалості процесу розробки ПЗ. Так, для визначення тривалості виконання 20 завдань необхідно близько 1 години у порівнянні з 3 годинами за стандартним підходом.

3.4 Висновки до розділу 3

Розроблено інформаційну технологію пошуку асоціативних правил щодо тривалості розробки програмного забезпечення, яка базується на запропонованій інформаційній моделі відповідного процесу. Використання даної інформаційної технології дозволить підвищити точність визначення тривалості виконання завдання розробником, а також скоротити тривалість пошуку асоціативних правил, які для цього використовуються. Спроековано UML – діаграму компонентів інформаційної технології пошуку асоціативних

правил щодо тривалості розробки програмного забезпечення.

Представлено процес розробки програмного забезпечення на різних рівнях деталізації, використовуючи єдиний математичний апарат, а саме ланцюги Маркова. Так, для розробки програмного забезпечення *PCounter* необхідно здійснити 192 кроки.

Промодельовано процес пошуку асоціативних правил щодо тривалості розробки програмного забезпечення з побудовою UML-діаграми діяльності даного процесу. Проведене моделювання показало зростання швидкості пошуку АП до 2% при кількості завдань більше 750. Також в результаті моделювання було отримано лінійну залежність між кількістю знайдених асоціативних правил та кількістю розроблених завдань у базі даних: кількість знайдених АП зростає в 10,9 разів, у випадку збільшення кількості завдань у базі даних в 20 разів. Також результати моделювання показали, що точність визначення тривалості виконання завдань зросла у 2,3 рази при збільшенні кількості знайдених асоціативних правил у 11 разів, що доводить доцільність пошуку таких залежностей щодо тривалості розробки програмного забезпечення.

Моделювання процесу використання знайдених асоціативних правил показало зменшення тривалості процесу визначення часу, необхідного на виконання завдання розробником певної кваліфікації, в 5 разів, у випадку коли кількість завдань рівна 20.

Основні результати наукового дослідження опубліковано в [35] – [38].

РОЗДІЛ 4

ІНФОРМАЦІЙНА СИСТЕМА ПОШУКУ АСОЦІАТИВНИХ ПРАВИЛ ЩОДО ТРИВАЛОСТІ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Для створення інформаційної системи на основі інформаційної технології пошуку асоціативних правил щодо тривалості розробки програмного забезпечення було:

- запропоновано структуру інформаційної системи, що розробляється;
- спроектовано базу даних розроблюваної інформаційної системи;
- написано програмний код.

Також оцінено точність визначення часу, необхідного на виконання завдання розробником з використанням знайдених асоціативних правил щодо тривалості розробки програмного забезпечення, що і дозволило оцінити розроблену інформаційну технологію.

4.1 Рекомендації щодо розробки інформаційної системи пошуку асоціативних правил щодо тривалості розробки програмного забезпечення

4.1.1 Структура інформаційної системи

Розроблена ІТ призначена для пошуку асоціативних правил щодо тривалості розробки програмного забезпечення з подальшим їх використанням для визначення часу, необхідного на виконання конкретного завдання розробником.

Для використання розробленої інформаційної технології потрібно розробити програмне забезпечення перед створенням якого спочатку потрібно спроектувати його структуру.

Обрана структура ПЗ повністю визначає такі його характеристики як надійність та зручність супроводу; впливає на зручність використання і

ефективність ПЗ. Для проектування структури потрібно враховувати можливі протиріччя між вимогами до різних характеристик і вміти вибирати компромісні рішення, що дають прийнятні значення за всіма показниками [147] – [149].

При розробці інформаційної системи потрібно враховувати, що: дані зберігаються в сховищі, яким у випадку розроблюваного ПЗ для пошуку асоціативних правил є база даних; потрібно виконувати операції їх завантаження у сховище, отримання зі сховища та відповідно запису до сховища змінених даних. База даних інформаційної системи пошуку АП щодо тривалості розробки ПЗ містить у собі таку інформацію:

- завдання, що були реалізовані під час розробки ПЗ та на основі яких здійснюватиметься пошук АП;
- розробники, що працюють у даній компанії;
- знайдені асоціативні правила;
- завдання, для яких визначений час їх реалізації розробником на основі знайдених асоціативних правил.

База даних є загальною для усіх користувачів розроблюваного ПЗ. Проте, для кожного користувача доступні дані лише тієї компанії, у якій вони працюють. Це реалізується за допомогою різних рівнів автентифікації.

Оскільки, розроблена на основі інформаційної технології пошуку асоціативних правил щодо тривалості розробки ПЗ інформаційна система застосовуватиметься користувачами локально, то потрібно обрати шаблон ПЗ, який дозволив би її зручне проектування та реалізацію. При проектуванні інформаційної системи пошуку АП щодо тривалості розробки ПЗ необхідно застосувати архітектурний шаблон Model-View-Controller (MVC) для розмежування алгоритмічної частини від інтерфейсу та забезпечення спільного використання компонентів. Шаблон – модель-представлення-контролер [150] дозволяє здійснити поділ основних елементів будь-якого програмного забезпечення: дані, інтерфейс користувача і керуюча логіка на три окремих компоненти:

- модель;
- представлення;
- контролер.

Такий поділ дозволяє модифікувати дані компоненти незалежно один від одного, що є дуже зручним щодо розробки ПЗ [151].

Модель (Model) відповідає за надання даних і методи роботи з ними. Вона реагує на команди контролера, змінюючи свій стан [152]. Модель може бути двох видів:

- пасивною, в такому випадку вона не знає про існування представлення і контролера. Контролер відстежує зміни моделі і оповіщає представлення;
- активною, тобто вона може самостійно оповістити представлення про зміну свого стану.

Для розробки ІС на основі розробленої ІТ буде використана активна модель, що міститиме в собі бізнес-логіку розроблюваного програмного забезпечення. Для уникнення порушення основної концепції MVC (незалежність моделі від представлення і контролера) та для реалізації механізму оповіщення використовуватиметься шаблон проектування Observer.

Даний шаблон проектування (Observer) доцільно використовувати щодо розробки програмного забезпечення пошуку асоціативних правил, тому що він визначає залежність «один до багатьох» між об'єктами програмного забезпечення. Тобто, у випадку змін стану одного об'єкта, всі залежні від нього об'єкти сповіщають про цю подію.

Контролер (Controller) в свою чергу інтерпретує дії користувача розробленої ІС, сповіщаючи модель про необхідність змін [153].

Основною задачею представлення (View) є отримання та відображення даних моделі користувачеві, реагуючи на зміни моделі [153]. Для ІС пошуку асоціативних правил такими даними будуть:

- знайдені асоціативні правила;

– визначений на їх основі час, необхідний для виконання завдання розробником певного рівня кваліфікації.

Клієнтська частина побудована з використанням набору інструментів JavaFX для створення інтернет-застосунків з можливістю підвантаження медіа та змісту, що не вимагають встановлення додаткових компонентів на комп'ютер. Дані, що передаються з клієнта на сервер, захищені від зовнішнього втручання та сторонньої модифікації за рахунок зберігання у закодованому вигляді.

На рис. 4.1 наведено діаграму прецедентів програмного забезпечення пошуку асоціативних правил щодо тривалості розробки ПЗ, що відображає множину акторів, прецедентів (варіантів використання) обмежених границею системи та асоціацій між акторами і прецедентами.

Діаграми прецедентів дозволяють представити проектоване програмне забезпечення у вигляді сутностей чи акторів, що взаємодіють із ПЗ за допомогою варіантів використання (прецедентів). Варіанти використання описують функціональні можливості, доступні актору [154].

Як видно із діаграми, основним «актором» виступає менеджер проекту, а варіантами використання системи є:

- завантаження даних для обробки;
- завантаження результатів обробки даних;
- класифікація даних;
- пошук асоціативних правил із класифікацією завдань та без їх класифікації;
- використання знайдених асоціативних правил.

З одного боку менеджер проекту є ініціатором усіх можливих варіантів використання, з іншого – лише він може отримати результати обробки даних. Між варіантом використання «класифікація даних» і «пошук АП з класифікацією завдань» існує тип зв'язку «асоціація», оскільки без класифікації неможливо здійснити відповідний тип пошуку АП. Також між прецедентом «завантаження даних для обробки» та усіма іншими

прецедентами крім «пошук АП з класифікацією завдань» та «завантаження результатів обробки даних» існує тип зв'язку «включення». Між варіантами використання «завантаження результатів обробки даних» та «пошук АП без класифікації завдань», «пошук АП із класифікацією завдань», «використання знайдених АП» існує зв'язок включення, тому що перед отриманням результатів роботи програмного забезпечення пошуку АП щодо тривалості розробки ПЗ потрібно виконати дії, що вказані вище.

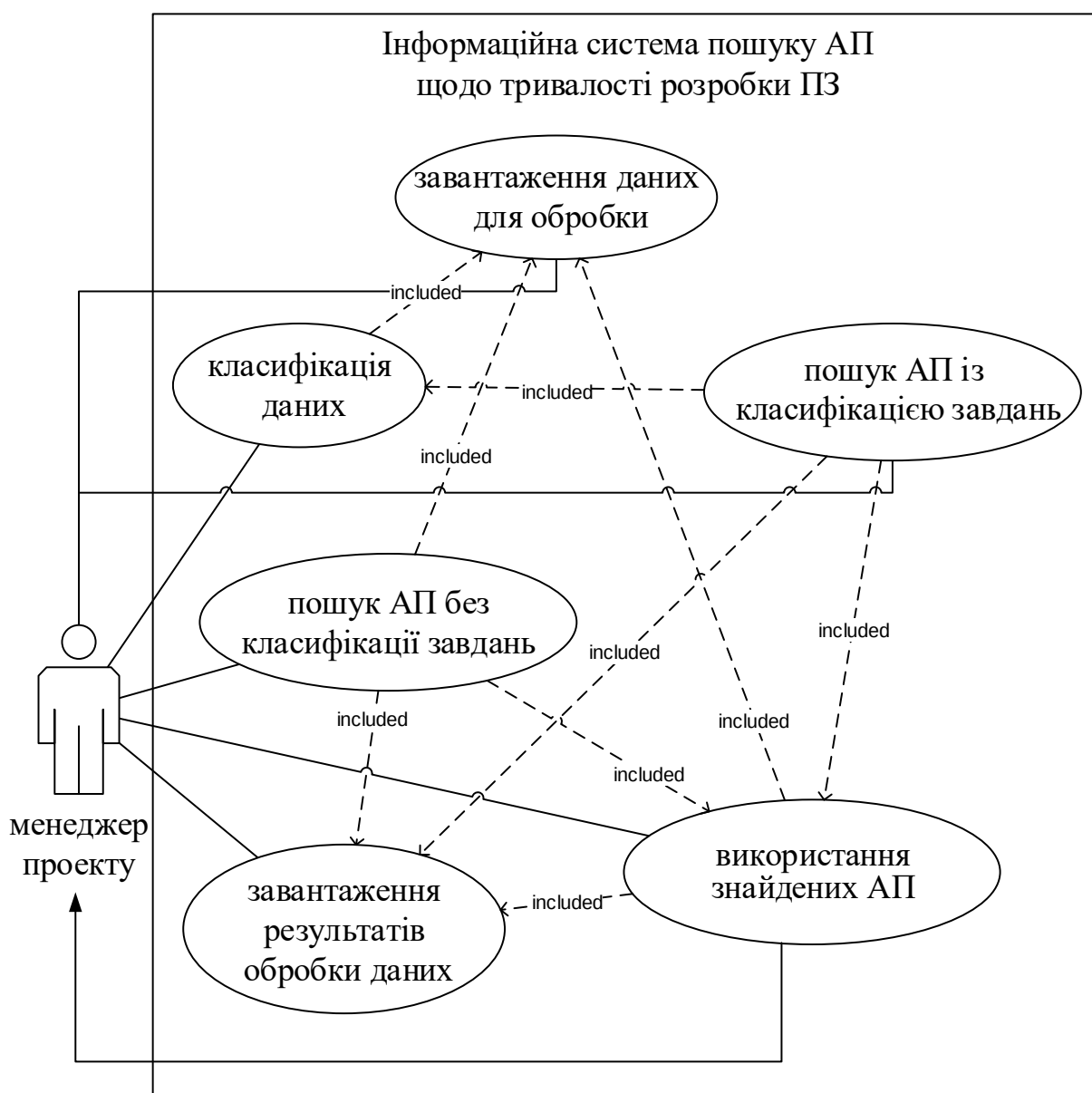


Рисунок 4.1 – Діаграма прецедентів використання інформаційної системи пошуку АП щодо тривалості розробки ПЗ

Відобразимо основні об'єкти програмного забезпечення пошуку асоціативних правил щодо тривалості розробки ПЗ, послідовності відправлених повідомлень між ними та взаємодії об'єктів, упорядкованих у часі за рахунок UML-діаграма послідовностей.

UML-діаграма послідовностей – це спосіб документування проекту з точки зору сценаріїв використання, оскільки вони уточнюють діаграму прецедентів за рахунок більш детального опису логіки сценаріїв використання. Діаграми послідовностей, зазвичай, містять об'єкти, які взаємодіють у рамках сценарію; повідомлення, якими вони обмінюються; результати, які пов'язані з повідомленнями [155].

UML-діаграму послідовностей програмного забезпечення пошуку асоціативних правил щодо тривалості розробки ПЗ наведено на рис. 4.2.

Користувач програмного забезпечення пошуку асоціативних правил щодо тривалості розробки ПЗ – це менеджер проекту.

Як видно із діаграми послідовностей, користувач не має прямого доступу до БД та виконує усі операції через відповідний графічний інтерфейс, який він також використовує для перегляду інформації.

Основними запитам користувача є:

- внесення завдань, виконаних при розробці програмного забезпечення до бази даних;
- пошук асоціативних правил серед збережених завдань;
- аналіз завдань, для яких потрібно визначити тривалість розробки розробником певної кваліфікації.

Результат, який користувач може отримати використовуючи програмне забезпечення пошуку асоціативних правил:

- знайдені асоціативні правила;
- завдання з визначеним часом, необхідним на їх виконання.

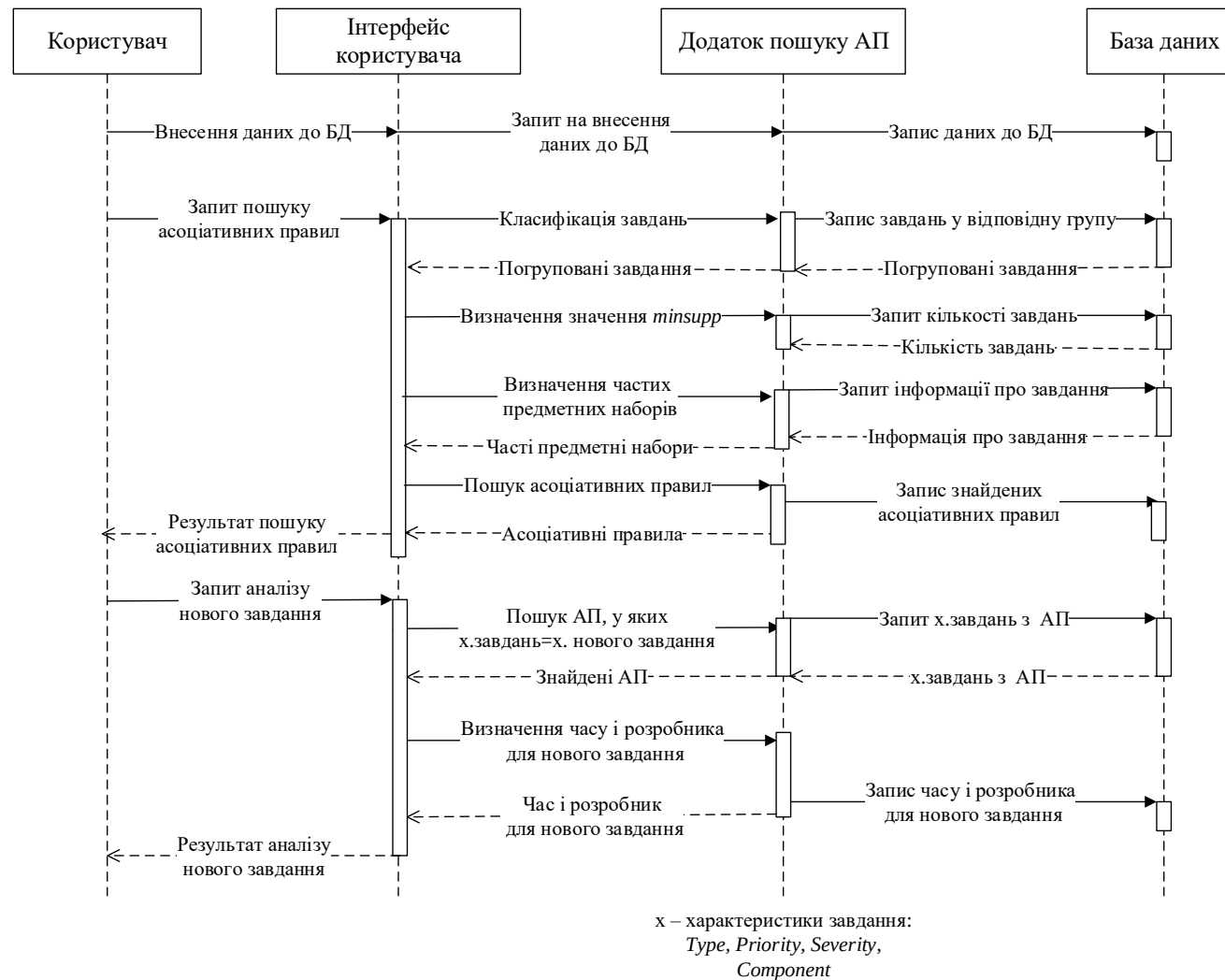


Рисунок 4.2 – UML-діаграма послідовностей програмного забезпечення пошуку асоціативних правил щодо тривалості розробки ПЗ

Графічний інтерфейс – це компонент, за допомогою якого користувач може здійснити ідентифікацію, автентифікацію та авторизацію в даному ПЗ, внести дані до бази даних, здійснити запит пошуку асоціативних правил чи аналізу нового завдання (визначення часу, необхідного для його виконання розробником із певним рівнем кваліфікації), отримати результати попередніх запитів.

Додаток програмного забезпечення для пошуку асоціативних правил – це основний компонент, в якому реалізується розроблена інформаційна технологія. Даний додаток напряду взаємодіє з базою даних. В ньому зосереджені основні модулі розроблюваного ПЗ:

- модуль навчання алгоритму класифікації;
- модуль класифікації завдань відповідно до їх характеристик;
- модуль обрахунку значення *minsupp*;
- модуль визначення частих предметних наборів;
- модуль пошуку асоціативних правил;
- модуль використання знайдених асоціативних правил;
- модуль формування результатів процесу пошуку АП, процесу визначення часу для нових завдань та збереження цієї інформації до БД.

У базі даних зберігаються класифіковані завдання, знайдені асоціативні правила щодо тривалості розробки програмного забезпечення та завдання з визначеним часом, необхідним на їх виконання.

Зв'язки між класами, інтерфейсами та об'єктами програмного забезпечення пошуку АП щодо тривалості розробки ПЗ відобразимо за рахунок UML діаграми класів. Вона дозволить представити статичну структуру системи за рахунок класів об'єктно-орієнтованого програмування [156], [157].

На рис. 4.3 наведено узагальнену UML діаграму класів, розроблюваного ПЗ для пошуку асоціативних правил щодо тривалості розробки програмного забезпечення.

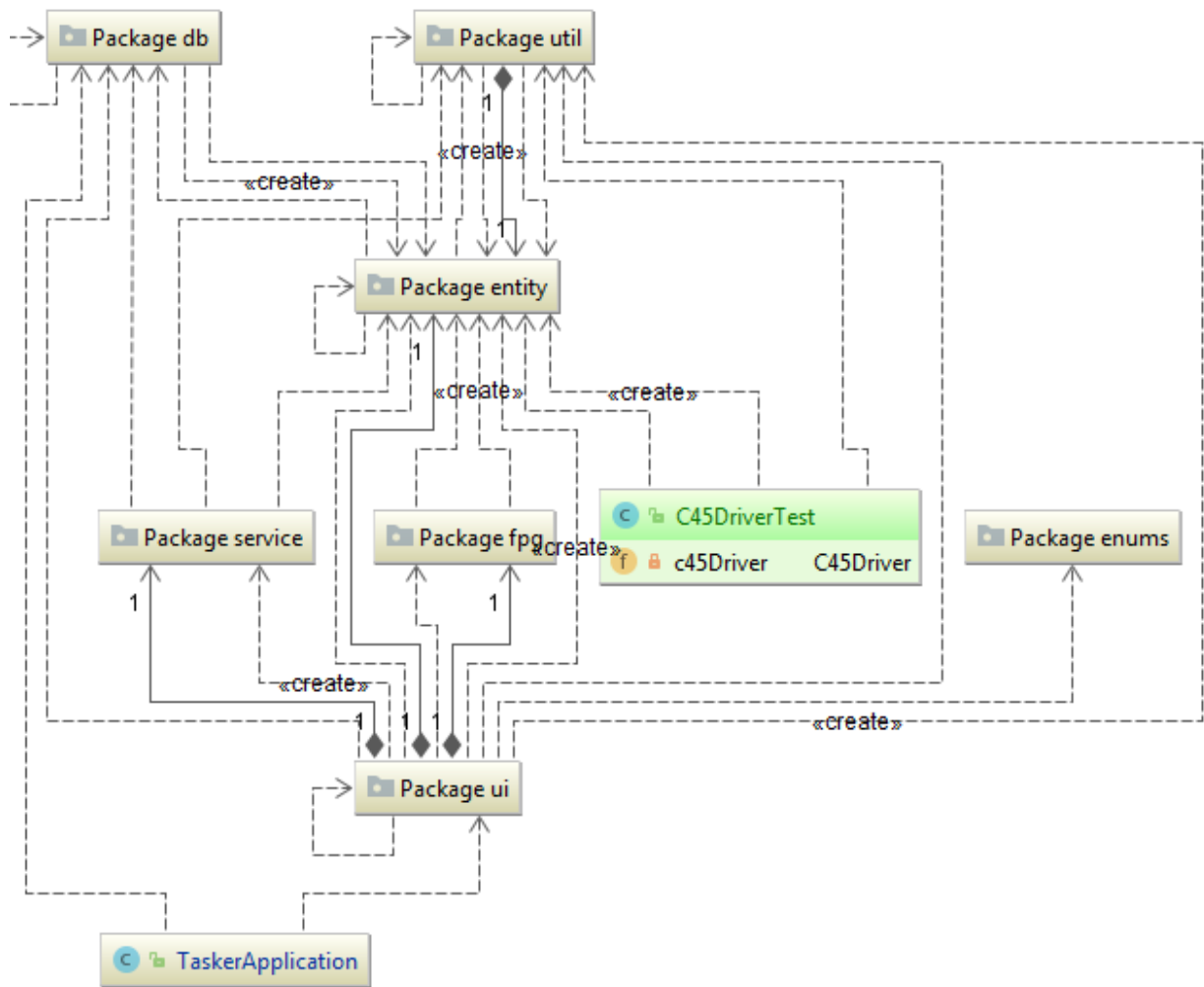


Рисунок 4.3 – Узагальнена UML-діаграма класів ПЗ для пошуку АП щодо тривалості розробки програмного забезпечення

На рис. 4.4 наведено UML діаграму класів графічного інтерфейсу, програмного забезпечення пошуку асоціативних правил щодо тривалості розробки програмного забезпечення.

Отже, було розроблено структуру інформаційної системи пошуку асоціативних правил щодо тривалості розробки програмного забезпечення, в основі якої лежить архітектурний шаблон Model-View-Controller, що дозволило розділити алгоритмічну частину від інтерфейсу, а також забезпечило спільне використання компонентів.

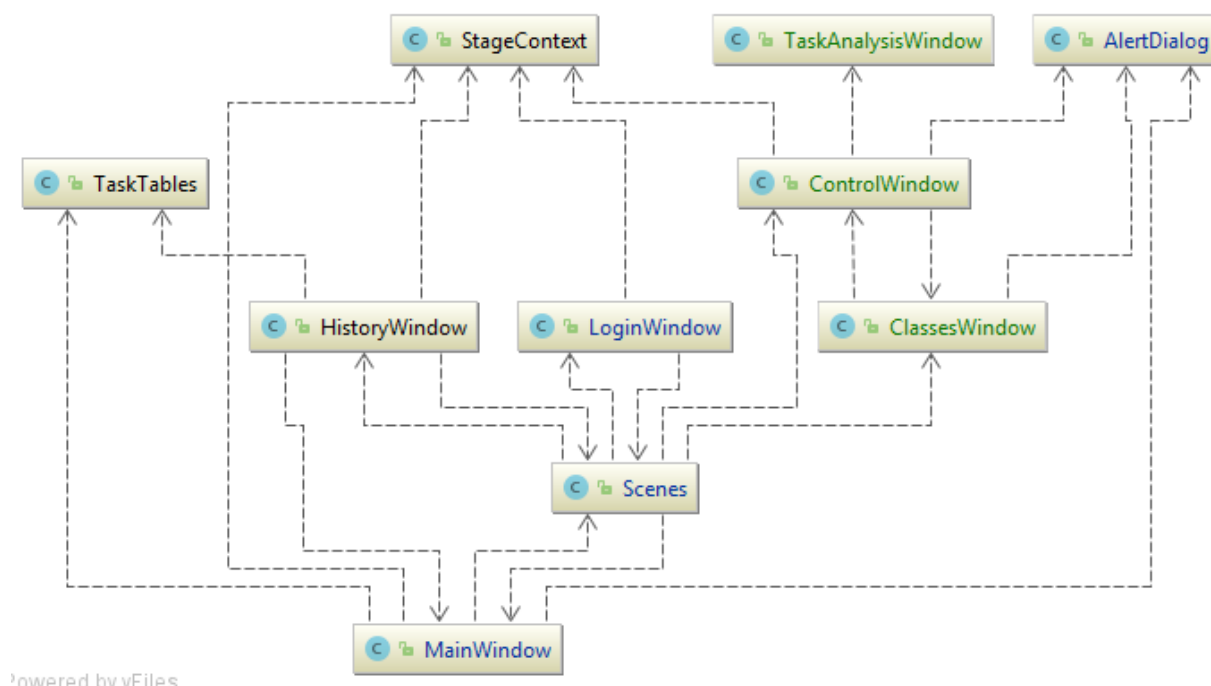


Рисунок 4.4 – UML-діаграма класів графічного інтерфейсу ПЗ пошуку асоціативних правил щодо тривалості розробки програмного забезпечення

Також визначено основні функціональні можливості даного програмного забезпечення та відображено їх порядок виконання за допомогою UML-діаграми послідовностей. Основні модулі ПЗ, що розробляється, відображено на схемі їх взаємодії, а потоки даних відображено за допомогою DFD діаграми. З використанням UML-діаграми класів було відображено класи, інтерфейси та об'єкти програмного забезпечення пошуку АП щодо тривалості розробки ПЗ а також зв'язки між ними.

4.1.2 База даних програмного забезпечення пошуку асоціативних правил щодо тривалості розробки

Для розробки інформаційної системи пошуку асоціативних правил щодо тривалості розробки ПЗ потрібно використовувати об'єктно-орієнтовне програмування, оскільки воно дозволить відобразити сукупності об'єктів та модулів, що є екземплярами певного класу, які, в свою чергу, утворюють ієрархію із властивістю унаслідування. Для реалізації даного підходу

потрібно використовувати мову програмування високого рівня. Оскільки інформаційна система пошуку асоціативних правил щодо тривалості розробки ПЗ складається з графічної та back end частин, необхідно вибрати таку мову програмування, яка б дозволила їх реалізацію без використання додаткових засобів програмування. Проаналізувавши означені вимоги, можна зробити висновок про доцільність застосування мови програмування Java для розробки додатка ПЗ пошуку асоціативних правил.

Для розробки інформаційної системи пошуку асоціативних правил щодо тривалості розробки ПЗ використано середовище програмування NetBeans IDE 8.2 оскільки воно є досить зручним для створення інтерфейсу інформаційної системи та використання ОО мови програмування Java.

Лістинг програмної реалізації основних модулів інформаційної системи пошуку АП щодо тривалості розробки ПЗ наведено в додатку Г. Інструкція користувача інформаційної системи пошуку АП щодо тривалості розробки ПЗ наведено в додатку Д.

Як було зазначено в підрозділі 4.1, БД – є важливим елементом інформаційної системи пошуку асоціативних правил щодо тривалості розробки ПЗ. Існуючі бази даних можна поділити на два класи: реляційні (SQL) та не реляційні (non-SQL), які відрізняються між собою проектуванням, типом даних, що підтримується, та способом зберігання інформації [158].

Реляційні бази даних відповідають вимогам атомарності, незаперечуваності, ізолюваності та довговічності, що дозволяє зменшити ймовірність незапланованої поведінки системи та забезпечити цілісність бази даних. У реляційних базах даних інформація структурована, і структура не піддається частковим змінам [159], [160].

Не реляційні БД дозволяють зберігати потужні обсяги неструктурованої інформації, без обмежень типу даних, що зберігаються. Більше того, при необхідності в процесі роботи можна додати нові типи

даних. Також такі БД дозволяють використовувати хмарні сховища для зберігання та обробки інформації [161].

Основними вимогами, які повинна задовольняти БД інформаційної системи пошуку асоціативних правил щодо тривалості розробки ПЗ, є:

- незалежність від фізичного рівня представлення;
- зручність і розуміння організації даних;
- максимальна гнучкість при обробці запитів;
- можливість розширення бази даних за рахунок приєднання нових елементів і записів, та не змінюючи при цьому існуючих підсхем та прикладних програм.

В результаті аналізу вимог та існуючих видів баз даних, можна зробити висновок, що реляційна модель даних є оптимальною для проектування бази даних інформаційної системи пошуку асоціативних правил щодо тривалості розробки ПЗ, оскільки вона забезпечує ефективне використання ресурсів системи шляхом мінімізації дублікатів. Реляційна модель даних є максимально прийнятною для даної предметної області тому що структура моделі повністю відповідає реляційним представленням.

СУБД MS SQL SERVER доцільно застосовувати для маніпулювання даними про пошук асоціативних правил щодо тривалості розробки ПЗ, оскільки вона зручна у користуванні, має високу надійність та має можливість роботи з ОО мовою програмування Java.

Графічне зображення ER-моделі бази даних інформаційної системи пошуку асоціативних правил щодо тривалості розробки ПЗ, з використанням нотації Чена, представлено на рис. 4.5.

Концептуальна схема даної БД відображає:

- основні сутності обраної предметної області: завдання, проект, асоціативні правила, компанія, розробник, менеджер проекту та нове завдання;
- зв'язки між сутностями, що відображено у ромбах;
- зв'язки між об'єктами сутностей: 1:1, Б:Б, 1:Б.

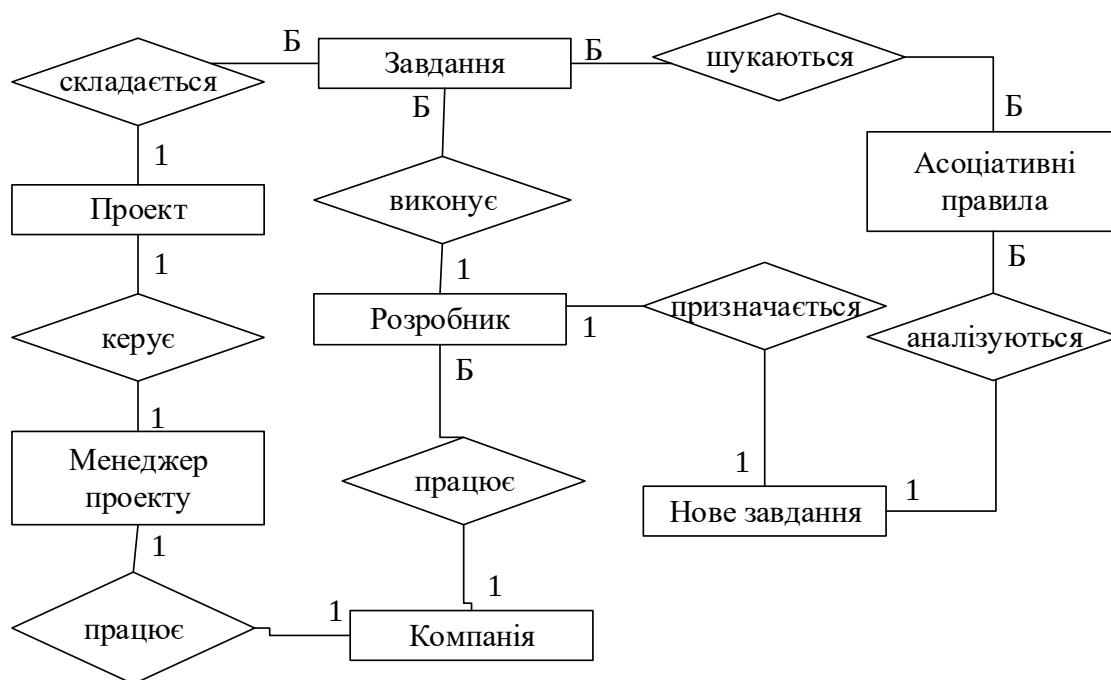


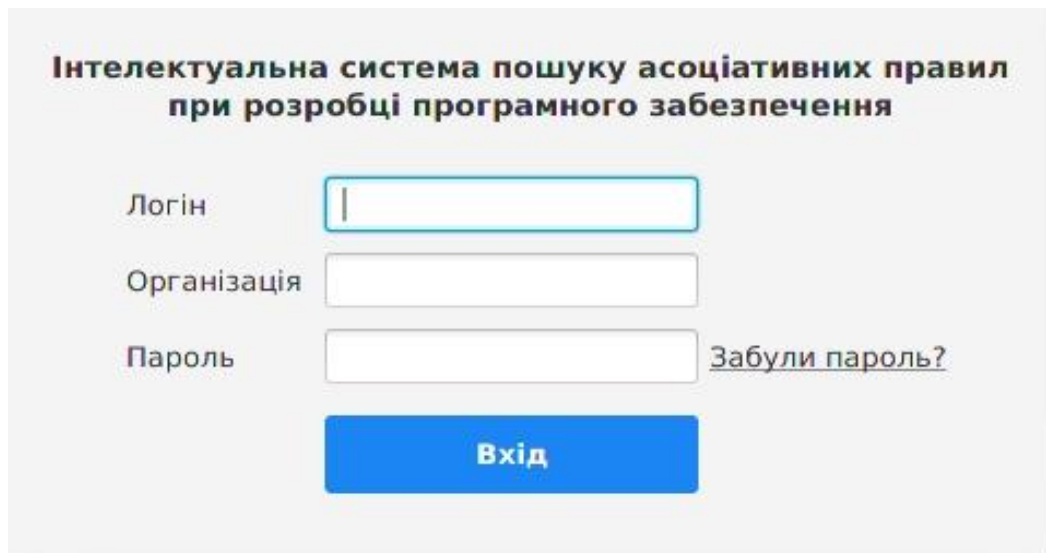
Рисунок 4.5 – ER-модель предметної області «Розробка програмного забезпечення»

Отже, просту та надійну у використанні об'єктно-орієнтовану мову програмування Java доцільно застосовувати для розробки інформаційної системи з використанням інформаційної технології пошуку асоціативних правил щодо тривалості розробки програмного забезпечення. Наведено реляційну базу даних розроблюваної інформаційної системи пошуку асоціативних правил щодо тривалості розробки програмного забезпечення та обрано СУБД MS SQL SERVER для зберігання інформації та надання швидкого доступу до неї.

4.1.3 Графічний інтерфейс користувача інформаційної системи пошуку асоціативних правил

Графічний інтерфейс користувача інформаційної системи пошуку асоціативних правил щодо тривалості розробки програмного забезпечення наведено на рис. 4.6 – 4.8. Інструкцію користувача даним програмним забезпеченням наведено у додатку Д.

Вікно автентифікації до розроблюваної ІС представлено на рис. 4.6. Для автентифікації користувач повинен ввести свій логін і пароль, а також вказати назву компанії, щоб мати доступ лише до завдань своєї організації.



The screenshot shows a login interface with the title "Інтелектуальна система пошуку асоціативних правил при розробці програмного забезпечення". It contains three input fields: "Логін" (Login), "Організація" (Organization), and "Пароль" (Password). The "Логін" field has a blue border and a cursor. The "Пароль" field has a "Забули пароль?" (Forgot password?) link next to it. Below the fields is a blue button labeled "Вхід" (Login).

Рисунок 4.6 – Вікно автентифікації до ПЗ пошуку асоціативних правил щодо тривалості розробки програмного забезпечення

Після логіну користувачу відображається головне вікно інформаційної системи, в якому зліва знаходиться меню з основними функціональними можливостями, а справа відображаються результати роботи ІС пошуку асоціативних правил щодо тривалості розробки програмного забезпечення (рис. 4.7).

Якщо АП для компанії та програмного забезпечення, що розробляється, вже були знайдені раніше, то після автентифікації вже не потрібно буде навчати алгоритм класифікації чи здійснювати пошук АП, а одразу можна їх використовувати для визначення часу, необхідного на реалізацію певного завдання, розробником певної кваліфікації.

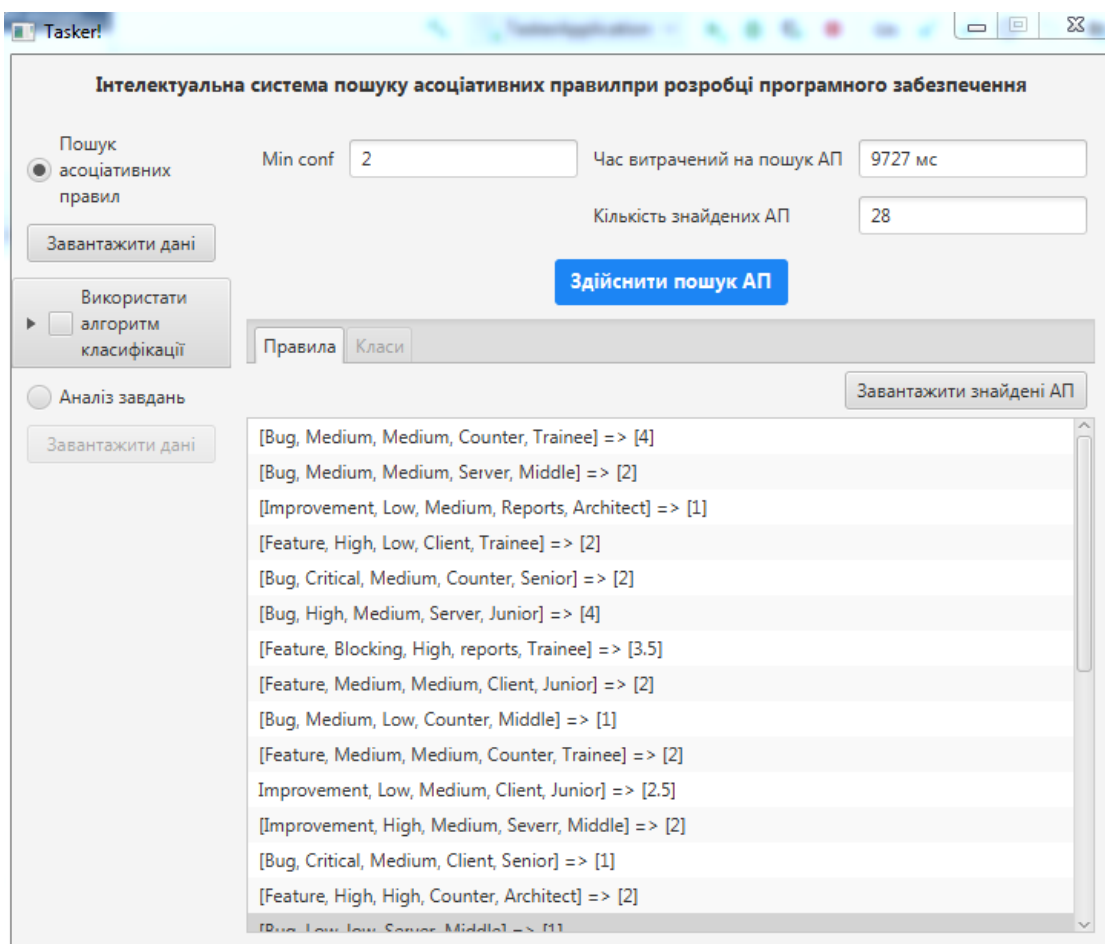


Рисунок 4.7 – Головне вікно інформаційної системи пошуку асоціативних правил щодо тривалості розробки ПЗ

Вікно з результатами класифікації завдань на три класи залежно від їх характеристик наведено на рис. 4.8.

Результати пошуку асоціативних правил, знайдених щодо тривалості розробки ПЗ, відображаються у вікні, представленою на рис. 4.7.

Усі результати роботи ІС пошуку асоціативних правил щодо тривалості розробки програмного забезпечення можна завантажити у вигляді xsl файлу для зручності їх аналізу та подальшого використання.

Рисунок 4.8 – Завдання, класифіковані програмним забезпеченням пошуку АП щодо тривалості розробки ПЗ

Отже, розроблено графічний інтерфейс, за допомогою якого користувачі взаємодіють з інформаційною системою пошуку асоціативних правил щодо тривалості розробки програмного забезпечення.

4.2 Результати впровадження інформаційної технології пошуку асоціативних правил щодо тривалості розробки програмного забезпечення

Оскільки використання методів інтелектуального аналізу даних, а саме методів пошуку асоціативних правил, щодо тривалості розробки програмного забезпечення дозволяє покращити даний процес, то потрібно оцінити результати використання розробленої інформаційної технології.

Дослідження результатів впровадження інформаційної технології пошуку асоціативних правил щодо тривалості розробки програмного

забезпечення полягає у здійсненні пошуку таких асоціативних залежностей з використанням удосконаленого методу FP-Growth, а також їх використання для визначення тривалості виконання завдання розробником певної кваліфікації.

Запропоновану інформаційну систему пошуку асоціативних правил щодо тривалості розробки програмного забезпечення, в основі якої лежить розроблена інформаційна технологія відповідного процесу, було впроваджено в ТОВ «Бізнес-Програми» у вигляді програмного забезпечення для пошуку асоціативних правил щодо тривалості розробки програмного забезпечення (додаток А). Використання розробленої інформаційної технології дозволило приймати ефективні управлінські рішення при плануванні та керуванні процесом розробки програмного забезпечення, оскільки збільшилася точність визначення часу, необхідного розробнику певної кваліфікації на виконання завдання. Це досягнуто за рахунок використання асоціативних правил щодо тривалості розробки ПЗ, терміни пошуку яких скоротилися.

Запропонована інформаційна система використовувалася при розробці та налаштуванні двох програмних продуктів: «BAS» та «FREDO» ТОВ «Бізнес-Програми».

Результати оцінювання точності визначення тривалості виконання завдання з використанням розробленої інформаційної технології пошуку асоціативних правил щодо тривалості розробки програмного забезпечення, для програм BAS та FREDO наведено у табл. 4.1.

Як видно з табл. 4.1 зі збільшенням кількості завдань в БД, серед яких здійснюється пошук АП, збільшується кількість асоціативних залежностей щодо тривалості розробки ПЗ. Таке зростання дозволило підвищити точність визначення тривалості виконання завдань:

а) для програмного забезпечення BAS в 1,7 рази (з 9,3% при 18 знайдених АП до 5,5% при знайдених 105 АП);

б) для програмного забезпечення FREDO в 1,6 рази (з 8,93% при 21 знайдених АП до 5,4% при знайдених 101 АП).

Таблиця 4.1 – Результати оцінювання точності визначення тривалості виконання завдання з використанням розробленої інформаційної технології

Програмне забезпечення	Кількість завдань в БД	Кількість знайдених АП	Точність визначення (%)
BAS	100	18	9,3
	200	40	8,6
	300	51	7,8
	400	67	7
	500	90	6,4
	600	105	5,5
FREDO	100	21	8,9
	200	39	8,6
	300	52	7,7
	400	67	6,5
	500	91	6,1
	600	101	5,4

Відповідно до даних, наданих менеджерами проектів, на початку тижня відбувався аналіз поточних завдань з метою визначення часу, необхідного на виконання цих завдання розробниками.

Результати використання розробленої інформаційної технології пошуку асоціативних правил щодо тривалості розробки програмного забезпечення, з метою визначення часу, необхідного на виконання завдання розробником певної кваліфікації, наведено у табл. 4.2.

Розраховане відношення Dur тривалості визначення часу, необхідного для розробки ПЗ, з використанням стандартного підходу, до тривалості визначення часу, необхідного для розробки ПЗ, із використанням запропонованого підходу, вказує у скільки разів скоротилася така тривалість. Це можна розрахувати використовуючи вираз виразі 4.1:

$$Dur = \frac{Ap_{stand}}{Ap_{prop}}, \quad (4.1)$$

де Ap_{stand} – тривалість визначення часу, необхідного для розробки ПЗ, з використанням стандартного підходу,

Ap_{prop} – тривалість визначення часу, необхідного для розробки ПЗ, з використанням запропонованого підходу.

Як видно з табл. 4.2 тривалість визначення часу, необхідного на виконання завдання розробником певної кваліфікації, скоротилась від 2 до 6 разів: для програмного забезпечення BAS тривалість визначення часу скоротилась від 2,75 до 6 разів, а для ПЗ FREDO – від 2 до 6 разів. Таке скорочення досягнуто за рахунок застосування запропонованого підходу використання знайдених асоціативних правил щодо розробки програмного забезпечення.

Таблиця 4.2 – Результати використання розробленої інформаційної технології пошуку асоціативних правил щодо тривалості розробки програмного забезпечення

Програмне забезпечення	Кількість завдань	Тривалість визначення часу, необхідного для розробки ПЗ (хв)		Dur
		Стандартний підхід	Запропонований підхід	
BAS	10	60	10	6
	12	110	40	2,75
	8	60	15	4
	11	110	25	4,4
	15	140	40	3,5
	10	30	10	3
FREDO	5	60	10	6
	7	60	10	6
	10	120	40	3
	8	90	15	6
	6	60	10	6
	7	40	20	2

На рис. 4.9 наведено графік залежності тривалості визначення часу, необхідного на виконання завдання розробником певної кваліфікації для ПЗ BAS.

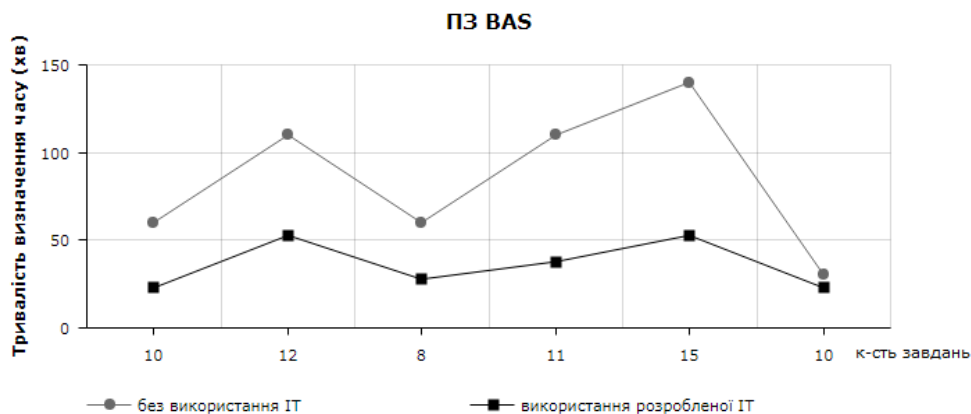


Рисунок 4.9 – Графік залежності тривалості визначення часу, необхідного на виконання завдання розробником певної кваліфікації для ПЗ BAS

Для деяких завдань не було знайдено відповідні асоціативні правила, тому визначення тривалості їх розробки здійснювалося опитуванням розробників, що збільшило тривалість даного процесу.

На рис. 4.10 наведено графік залежності тривалості визначення часу, необхідного на виконання завдання розробником певної кваліфікації для ПЗ FREDO.

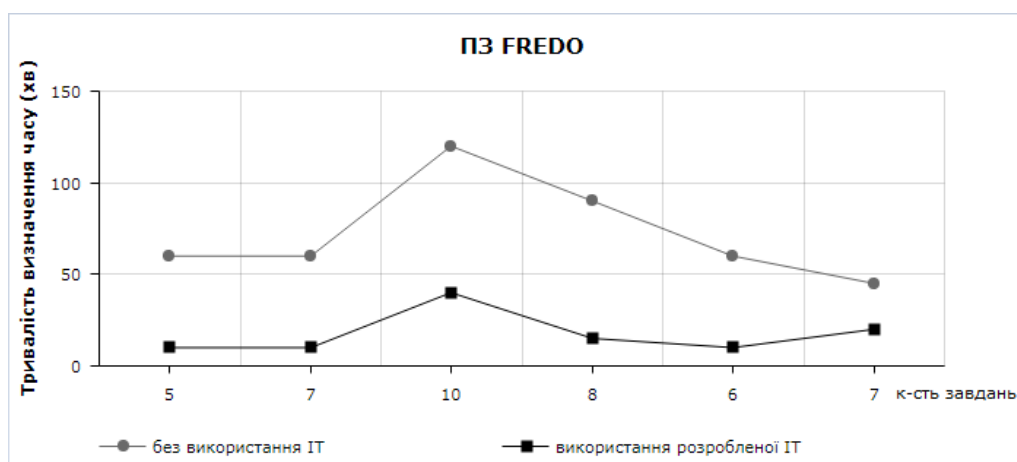


Рисунок 4.10 – Графік залежності тривалості визначення часу, необхідного на виконання завдання розробником певної кваліфікації для ПЗ FREDO

Отже, проаналізувавши результати, отримані при впровадженні інформаційної технології пошуку асоціативних правил щодо тривалості розробки програмного забезпечення при розробці програм BAS та FREDO, можна зробити висновок про доцільність застосування запропонованої технології під час розробки ПЗ. Вона дозволила збільшити точність визначення тривалості виконання завдання в 1,7 рази при збільшенні знайдених асоціативних правил до 100. Також використання розробленої інформаційної системи скоротило тривалість визначення часу, необхідного на виконання завдання розробником, від 2 до 6 разів в залежності від кількості знайдених АП та кількості завдань, для яких здійснюється дане визначення.

4.3 Висновки до розділу 4

Під час практичної реалізації інформаційної технології пошуку асоціативних правил щодо тривалості розробки ПЗ було запропоновано структуру відповідної інформаційної системи та за допомогою UML-діаграми послідовностей відображено її основні функціональні можливості. Наведено можливі потоки даних, що з'являються при використанні інформаційної системи пошуку асоціативних правил щодо розробки ПЗ. Зв'язки між класами програмного забезпечення пошуку асоціативних правил щодо тривалості розробки ПЗ відображено з використанням UML-діаграми класів.

Запропоновано реляційну базу даних інформаційної системи пошуку асоціативних правил щодо розробки ПЗ, для якої використовуватиметься СУБД MS SQL SERVER для зберігання інформації та можливості роботи з нею. Розроблено графічний інтерфейс, який дозволить користувачам взаємодіяти з розробленою інформаційною системою.

Розроблену інформаційну систему пошуку асоціативних правил щодо

тривалості розробки програмного забезпечення впроваджено на фірмі ТОВ «Бізнес-Програми», а доцільність її використання підтверджено експериментально. Точність визначення тривалості виконання завдання розробником певної кваліфікації зросла в 1,7 рази, за рахунок використання знайдених асоціативних правил. А швидкість визначення часу, необхідного на виконання завдання розробником певної кваліфікації, підвищилась до 6 разів у випадку застосування розробленої інформаційної системи пошуку і використання знайдених асоціативних правил щодо тривалості розробки програмного забезпечення.

Основні результати наукового дослідження опубліковано в [39].

ВИСНОВКИ

У дисертаційному дослідженні проведено теоретичне дослідження і наведено нове вирішення актуальної наукової задачі, яка полягає у підвищенні точності визначення часу необхідного на виконання завдання розробником, а також скороченні часу пошуку асоціативних правил щодо тривалості розробки програмного забезпечення, які для цього використовуються, шляхом розробки і впровадження відповідної інформаційної технології.

Основні наукові і практичні результати роботи:

1. Проаналізувавши сучасні засоби, що використовуються для планування та управління процесом розробки програмного забезпечення, було встановлено, що за допомогою них не можна прогнозувати тривалість виконання завдання чи проекту в цілому і, тому необхідно розробити інформаційну технологію пошуку асоціативних правил щодо тривалості розробки програмного забезпечення, яка б дозволила збільшити точність визначення тривалості виконання завдання розробником певної кваліфікації та скоротити тривалість пошуку асоціативних правил щодо тривалості розробки програмного забезпечення.

Аналіз сучасних моделей процесу розробки програмного забезпечення показав, що у них враховується лише така характеристика завдань як пріоритет. А тому існує необхідність у розробці інформаційної моделі, яка, на відміну від існуючих, враховувала б такі характеристики завдань (завдань, які потрібно вирішити при розробці програмного забезпечення) як: складність, пріоритет, критичність та тривалість їх виконання. Кожна із розглянутих моделей складається із основних етапів (вимоги до документації, проектування, аналіз, кодування, перевірка та тестування), тривалість яких можна визначити з використанням відповідних методів інтелектуального аналізу даних.

Розглянувши стан розвитку процесу розробки програмного забезпечення, було встановлено, що сучасні методи інтелектуального аналізу

даних широко використовуються під час даного процесу. Тому варто застосовувати метод визначення частих предметних наборів для пошуку асоціативних правил FP-Growth щодо тривалості розробки програмного забезпечення. Даний метод потрібно удосконалити з метою виправлення існуючих недоліків та адаптації до відповідної предметної області.

2. Для визначення характеристик завдань, що впливають на тривалість розробки завдань розраховано коефіцієнт Чупрова і на основі отриманих даних розроблено інформаційну модель процесу пошуку асоціативних правил щодо тривалості розробки програмного забезпечення. Встановлено, що тип, складність, пріоритет та критичність завдання, рівень кваліфікації розробника, а також компонент програмного забезпечення, що розробляється впливають на тривалість розробки завдання. Запропонована інформаційна модель, на відміну від існуючих, враховує наступні характеристики завдань: складність, пріоритет, критичність та тривалість їх виконання. Для розробленої інформаційної моделі вхідними даними є множина завдань, серед таких завдань буде здійснюватися пошук асоціативних правил з використанням удосконаленого методу FP-Growth; вихідними даними є множина завдань з визначеним часом, необхідним на їх реалізацію розробником певної кваліфікації. Час реалізації визначається за рахунок застосування знайдених асоціативних правил.

3. Удосконалено метод визначення частих предметних наборів для пошуку асоціативних правил FP-Growth, який, на відміну від існуючих, передбачає етап класифікації завдань на три класи, залежно від характеристик завдань. Розроблено алгоритм використання таких асоціативних правил для визначення часу, необхідного на виконання завдання розробником з певним рівнем кваліфікації, що допоможе менеджерам проектів при плануванні та управлінні процесом розробки програмного забезпечення.

4. Розроблено інформаційну технологію пошуку асоціативних правил щодо тривалості розробки програмного забезпечення в основі якої, на

відміну від існуючих, лежить удосконалений метод визначення частих предметних наборів для пошуку асоціативних правил FP-Growth та проведено моделювання такого процесу. Результати моделювання показали, що точність визначення тривалості виконання завдань зросла у 2,3 рази при збільшенні кількості знайдених асоціативних правил у 11 разів; тривалість пошуку асоціативних правил скоротилася на 2%, якщо кількість завдань понад 750; кількість знайдених асоціативних правил зростає в 10,9 разів, у випадку збільшення кількості завдань у базі даних в 20 разів.

5. На основі запропонованої інформаційної технології розроблено інформаційну систему пошуку асоціативних правил щодо тривалості розробки програмного забезпечення. Аналіз результатів впровадження запропонованої інформаційної технології та інформаційної системи у ТОВ «Бізнес-Програми», показав що: точність визначення тривалості виконання завдання розробником певної кваліфікації зросла в 1,7 рази за рахунок використання знайдених асоціативних правил; швидкість визначення часу, необхідного на виконання завдання розробником певної кваліфікації, підвищилась до 6 разів в залежності від кількості завдань, для яких цей час визначався. Це дало можливість приймати ефективні управлінські рішення при плануванні та керуванні процесом розробки програмного забезпечення.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

- [1] S. Konrad, “Requirements engineering in the development of large-scale systems” in *16th IEEE International Requirements Engineering Conference*, Barcelona, 2008, pp. 217 – 222.
- [2] R. S. Pressman, *Software Engineering: A Practitioner's Approach*, NY, USA: McGraw-Hill, 2014.
- [3] L. Chung, B. A. Nixon, E. Yu, and J. Mylopoulos, *Non-Functional Requirements in Software Engineering*, NY, USA: Springer US, 2000.
- [4] K. Wiegers, *Software Requirements 3rd Edition*, NY, USA: Microsoft Press, 2013.
- [5] S. J. Greenspan, and C. L. McGowan, “Structuring software development for reliability”, *Microelectronics Reliability*, vol. 17, no. 1, pp. 75 – 83, 1987.
- [6] G. Metkalf, *Lean Software Development: Avoiding project mishaps: An introduction*, Michigan, USA: Independently published, 2018.
- [7] G. E. Kaiser, P. H. Feiler, and S. S. Popovich, “Intelligent assistance for software development and maintenance”, *IEEE Software*, vol. 5, no. 3, pp. 40 – 49, 1988.
- [8] G. Piatetsky-Shapiro, and W. Frawley, *Knowledge Discovery in Databases*, CA, USA: MIT Press, 1991.
- [9] W. Zou, X. Xia, W. Zhang, Z. Chen, “An Empirical Study of Bug Fixing Rate”, in *IEEE 39th Annual Computer Software and Applications Conference*, Taichung, 2015, pp. 125 – 136.
- [10] F. Zhang, F. Khomh, Y. Zou, and A. Hassan, “An Empirical Study on Factors Impacting Bug Fixing Time”, in *19th Working Conference on Reverse Engineering*, Washington, 2012, pp. 225 – 234.
- [11] H. Zhang, L. Gong, and S. Versteeg, “Predicting Bug-Fixing Time: An Empirical Study of Commercial Software Projects”, in *35th International Conference on Software Engineering*, San Francisco, 2013, pp. 1042 – 1051.

- [12] P. Bhattacharya, and I. Neamtiu, “Bug-fix Time Prediction Models: Can We Do Better?” *MSR '11 Proceedings of the 8th Working Conference on Mining Software Repositories*, Honolulu, 2011, pp. 207 – 210.
- [13] A. Lamkanf, and S. Demeyer, “Filtering Bug Reports for Fix-Time Analysis”, *16th European Conference on Software Maintenance and Reengineering*, Szeged, 2012, pp. 379 – 384.
- [14] E. Giger, M. Pinzger, and H. Gall, “Predicting the Fix Time of Bugs”, *RSSE '10: Proceedings of the 2nd International Workshop on Recommendation System for Software Engineering*, NY, 2010, pp. 52 – 56.
- [15] А. А. Молчанов, *Моделирование и проектирование сложных систем*. Київ, Україна: Вища школа, 1988.
- [16] В. И. Скурихин и др. *Математическое моделирование*. Київ, Україна: Техніка, 1983.
- [17] Н. П. Бусленко. *Моделирование сложных систем*. Москва, Россия: Наука, 1978.
- [18] В. М. Трояновский. *Математическое моделирование в менеджменте*. Москва, Россия: РДЛ, 2002.
- [19] Р. Н. Кветний, І. В. Богач, О. Р. Бойко, та О. Ю. Софіна, *Комп'ютерне моделювання систем та процесів. Методи обчислень*. Вінниця, Україна: ВНТУ, 2012.
- [20] В. М. Дубовой, *Моделювання систем контролю та керування. Навчальний посібник*. Вінниця, Україна: ВНТУ, 2005.
- [21] Б. І. Мокін, та А. В. Камінський, *Математичне та комп'ютерне моделювання процесів оптимізації центрального електричних мереж*. Вінниця, Україна: УНІВЕРСУМ, 2005.
- [22] В. Б. Мокін, та Ю. С. Гавриков, “Новий підхід до двовимірного моделювання динаміки змішування річкових вод зі стічними, що надходять з берега”, *Вісник Херсонського національного технічного університету*, № 2, с. 340 – 343, 2002.

[23] О. В. Бісікало, “Класифікація образного пошуку та моделювання інсайту”, *Вісник Сумського державного університету. Серія Технічні науки*. №2, с.53 – 59, 2008.

[24] Д. С. Штобва, та А. А. Яковенко, “Прогнозування трудомісткості розробки програмних систем за допомогою нечіткої гібридної моделі” *Наукові праці ВНТУ. Серія Інформаційні технології та компютерна техніка*, №1, с. 1 – 7, 2014.

[25] T. O. Savchuk, and N. V. Pryimak, “Feasibility study of using associative rules during the software development”, on *IX international Conference of the Georgian Mathematical Union*, Batumi, 2018, pp. 203 – 204.

[26] Т. О. Савчук, та Н. В. Приймак, “Використання fpg-алгоритму для пошуку асоціативних правил при прийнятті рішень в управлінні процесами”, на *XLV НТК проф.-викл. складу, співробітників та студентів університету*, Вінниця, 2016, с. 929 – 930.

[27] T. O. Savchuk, N. V. Pryimak, A. Assembay, T. Zyska, M. Junisbekov, and A. Annabaev “The technology of searching the associative rules while developing the software”, *Proc. SPIE 10445, Photonics Applications in Astronomy, Communications, Industry, and High Energy Physics Experiments*, 2017, doi: 10.1117/12.2280900.

[28] Т. О. Савчук, та Н. В. Приймак, “Використання методів інтелектуального аналізу даних при тестуванні програмного забезпечення”, на *X міжнародній конференції «Інтернет-освіта-наука-2016»*, Вінниця, 2016, с. 28 – 29.

[29] Т. О. Савчук, та Н. В. Приймак, “Використання методів штучного інтелекту для прогнозування часу для виправлення дефектів розробки програмного забезпечення”, на *X міжнародній конференції «Інтернет-освіта-наука-2016»*, Вінниця, 2016, с. 30 – 31.

[30] Т. О. Савчук, та Н. В. Приймак, “Обґрунтування доцільності використання методу fp-growth (fpg) при розробці програмного

забезпечення”, *Вчені записки ТНУ імені В.І. Вернадського. Серія Технічні науки*, № 5, с. 41 – 45, 2018.

[31] Т. О. Савчук, та Н. В. Приймак, “Обґрунтування вибору методу визначення частих предметних наборів для пошуку асоціативних правил при розробці програмного забезпечення”, на *XI міжнародній конференції «Інтернет-освіта-наука-2018»*, Вінниця, 2018, с. 45 – 46.

[32] Т. О. Савчук, та Н. В. Приймак, “Інформаційна модель процесу пошук і використання асоціативних правил при розробці програмного забезпечення”, на *XI міжнародній конференції «Інтернет-освіта-наука-2018»*, Вінниця, 2018, с. 43 – 44.

[33] Т. О. Савчук, та Н. В. Приймак, “Розробка інформаційної моделі процесу пошуку асоціативних правил при розробці програмного забезпечення”, *Інформаційні технології та комп’ютерна інженерія*, № 2 (42), с. 43 – 48, 2018.

[34] Т. О. Савчук, та Н. В. Приймак, “Метод пошуку асоціативних правил при розробці програмного забезпечення, що базується на попередньому розбитті множини завдань” на *XLVIII НТК проф.-викл. складу, співробітників та студентів університету*, Вінниця, 2019, с. 655 – 656.

[35] T. O. Savchuk, and N. V. Pryimak, “Modeling of software development process with the Markov processes”, *Eastern-European journal of Enterprise Technologies . Information technology*, vol. 3, no. 2, pp. 33 – 38, 2017.

[36] Т. О. Савчук, та Н. В. Приймак, “Використання Марковських ланцюгів при моделюванні процесу розробки програмного забезпечення”, на *XLVI НТК проф.-викл. складу, співробітників та студентів університету*, Вінниця, 2017, с. 1019 – 1020.

[37] Т. О. Савчук, та Н. В. Приймак, “Розробка інформаційної технології процесу пошуку асоціативних правил при розробці програмного забезпечення”, *Наукові праці: наук. журн. / Чорном. нац. ун-т ім. П. Могили. Комп’ютерні технології*, вип. 305, т. 317, с. 6 – 11, 2018.

[38] Т. О. Савчук, та Н. В. Приймак, “Інформаційна технологія пошуку асоціативних правил при розробці програмного забезпечення”, на *Всеукраїнській наук.-практ. конф. здобувачів вищої освіти й молодих учених*, Кропивницький, 2018, с. 100 – 102.

[39] Т. О. Савчук, та Н. В. Приймак, “Інформаційна система пошуку асоціативних правил при розробці програмного забезпечення”, *Свідомство про реєстрацію авторського права на твір №82173*, Жов. 11, 2018.

[40] В. К. Батоврин, *Толковый словарь по системной и программной инженерии*. Москва, Россия: ДМК Пресс, 2012.

[41] IEEE Std 829 – 2008 IEEE Standard for Software and System Test Documentation.

[42] A. Stellman, and J. Greene, *Applied Software Project Management*. California, USA: O'Reilly Media, 2005.

[43] M. El Bajta, A. Idri, J. N. Ros, and J. L. Fernández-Alemán, “Software project management approaches for global software development: a systematic mapping study”, *Tsinghua Science and Technology*, vol. 23, no. 6, pp. 690 – 714, 2018.

[44] Ch. Chapman, and St. Warn, *Project risk management: processes, techniques and insights*. Chichester, UK: John Wiley, 1996.

[45] D. White, and J. Fortune, “Current practice in project management – an empirical study”, *International Journal of Project Management*, vol. 20, no. 1, pp. 1 – 11, 2002.

[46] Сетевые методы планирования и управления, 2019. [Электронный ресурс]. Доступно: <http://arprime.ru/planirovanie/setevoye-planirovaniye-i-upravleniye>.

[47] A. Wren, *Project Management A-Z: A Compendium of Project Management Techniques and How to Use Them*, London, UK: Routledge, 2003.

[48] В. Д. Секерин, Р. М. Нижегородцева, А. Е. Горохова, и Д. В. Секерин. *Инновации в маркетинге*. Москва, Россия: МГУИЭ, 2013.

- [49] Поиск критического пути модели, 2019. [Электронный ресурс]. Доступно: <http://projectimo.ru/planirovanie-proekta/kriticheskij-put.html>.
- [50] T. J. Havranek, *Modern Project Management Techniques for the Environmental Remediation Industry*, NY, USA: Routledge, 2017.
- [51] F. Harrison, and D. Lock, *Advanced Project Management A Structured Approach*, London, UK: Routledge, 2017.
- [52] 6 Benefits of Network Diagrams, 2019. [Online]. Available: <https://blog.masterofproject.com/network-diagrams/>.
- [53] Как диаграммы Ганта упрощают работу с проектами, 2019. [Электронный ресурс]. Доступно: <https://habr.com/ru/company/blog/415271/>.
- [54] Microsoft Project, 2019. [Online]. Available: <https://products.office.com/fr/project/project-and-portfolio-management-software?>.
- [55] Jira, 2019. [Online]. Available: <https://www.atlassian.com/software/jira>.
- [56] Sinnaps, 2019. [Online]. Available: <https://www.sinnaps.com/en/product>.
- [57] Optical tools. WBS Schedule Pro, 2019. [Online]. Available: <http://criticaltools.com/Network%20Charts.html>.
- [58] Organaza, 2019. [Online]. Available: <https://organaza.com/features/>.
- [59] 43 полезных сервиса для управления проектами. Без эпитетов, 2019. [Электронный ресурс]. Доступно: <https://habr.com/ru/post/276873/>.
- [60] Acunote, 2019. [Online]. Available: <https://www.acunote.com/>.
- [61] Application Development, 2019. [Online]. Available: <http://www.bestpricecomputers.co.uk/glossary/application-development.html>.
- [62] S. Wang, D. Samadhiya, and D. Chen, “Software Development and Quality Problems and Solutions by TRIZ”, in *International Symposium on Frontiers in Ambient and Mobile Systems*, MI, 2011, pp. 749 – 753.
- [63] Ещё раз про семь основных методологий разработки, 2019. [Электронный ресурс]. Доступно: <https://habr.com/company/edison/blog/2697/>.

[64] 12 best software development methodologies with pros and cons, 2019. [Online]. Available: <https://acodez.in/12-best-software-development-methodologies-pros-cons/>

[65] Waterfall Methodology in Project Management, 2019. [Online]. Available: <https://www.projectmanager.com/software/use-cases/waterfall>.

[66] What is V-model- advantages, disadvantages and when to use it?, 2019. [Online]. Available: <http://tryqa.com/what-is-v-model-advantages-disadvantages-and-when-to-use-it/>.

[67] What is Incremental model- advantages, disadvantages and when to use it?, 2019. [Online]. Available: <http://tryqa.com/what-is-incremental-model-advantages-disadvantages-and-when-to-use-it/>.

[68] What is iterative model?, 2019. [Online]. Available: <http://www.professionalqa.com/iterative-model>.

[69] Е. А. Мирошниченко, *Технология программирования*. Томск, Россия: ТПУ, 2008.

[70] R. Murch, *The Software Development Lifecycle – A Complete Guide*, Amazon Digital Services LLC, 2012.

[71] Разработка программного обеспечения: этапы и принципы, 2019. [Электронный ресурс]. Доступно: <https://habr.com/ru/company/blog/267671/>.

[72] K. Roebuck, *Systems Development Life Cycle (SDLC): High-impact Strategies - What You Need to Know: Definitions, Adoptions, Impact, Benefits, Maturity, Vendors*, TN, USA: Lightning Source, 2011.

[73] R. L. Glass, *Facts and Fallacies of Software Engineering*. Boston, USA: Addison-Wesley Professional, 2002.

[74] M. Halkidi, and G. Tsatsaronis. “Data mining in software engineering”, *Intelligent Data Analysis*, pp. 413 – 441, 2011.

[75] T. Xie, “Data Mining for Software Engineering”, *Computer*, vol. 42, no. 7, pp. 55 – 62, 2009.

[76] Q. Taylor, and Ch. Giraud-Carrier, “Applications of data mining in software engineering”, *Data Analysis Techniques and Strategies*, vol. 2, no. 3, pp. 243 – 257, 2010.

[77] А. А. Барсеагян, М. С. Куприянов, В. В. Степаненко, та И. И. Холод. *Технологии анализа данных: Data Mining, Visual Mining, text minig, OLAP*, Санкт-Петербург, Россия: БХВ-Петербург, 2007.

[78] Лекция 3: Методы и стадии Data Mining, 2019. [Электронный ресурс]. Доступно: <https://www.intuit.ru/studies/courses/6/6/lecture/162?page=2>.

[79] И. А. Чубукова, *Data Mining. Курс лекций интернет-университета INTUIT*, 2006.

[80] J. Anvik, “Automating bug report assignment”, in *Proceedings of the 28th International Conference on Software Engineering*, pp. 937 – 940, 2006.

[81] G. Canfora, and L. Cerulo, “Impact analysis by mining software and change request repositories”, in *Proceedings of the 11th IEEE International Software Metrics Symposium*, 2005.

[82] M. Christodorescu, S. Jha, and C. Kruegel, “Mining specifications of malicious behavior”, in *Proceedings of the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering*, pp. 5 – 14, 2007.

[83] W. Dickinson, D. Leon, and A. Podgurski, “Finding failures by cluster analysis of execution profiles”, in *Proceedings of the 23rd International Conference on Software Engineering*, pp. 339 – 348, 2001.

[84] J. S. Shirabad, T. C. Lethbridge, and S. Matwin, “Supporting software maintenance by mining software update records”, in *Proceedings of the IEEE International Conference on Software Maintenance*, pp. 22 – 31, 2001.

[85] D. Hand, H. Mannila, and P. Smyth, *Principles of data mining (adaptive computation and machine learning)*, Cambridge, UK: The MIT Press, 2001.

[86] M. Sharma, and M. Kumari. “Bug Assignee Prediction Using Association Rule Mining”, *ICCSA 2015. Lecture Notes in Computer Science*, pp. 444 – 457, 2015.

[87] A. Berson, and S. Smith, *Data Warehousing, Data Mining, and OLAP*, New York, USA: McGraw-Hill Education, 2007.

[88] Q. uz Zaman, A. Nadeem, and M. A. Sindhu, Formalizing a Use Case to a Kripke Structure, *Proceedings of the IASTED International Symposium Software Engineering and Applications*, 2015, pp. 232 – 239.

[89] C. Stirling, “Modal and temporal logics”, Great Britain: University of Edinburgh, Department of Computer Science, 1991.

[90] M. Sindhu, “Algorithms and Tools for Learning-based Testing of Reactive Systems”, PhD thesis, KTH Royal Institute of Technology, Stockholm, 2013.

[91] С. Синицын, и Н. Налютин. *Верификация программного обеспечения*. Москва, Россия: МИФИ, 2006.

[92] D. Dranidis, K. Tigka, and P. Kefalas, “Formal modelling of use cases with X-machines”, *Proceedings of the 1st South-East European Workshop on Formal Method*, no. 3, pp. 72 – 83, 2003.

[93] User Story vs Use Case, 2019. [Online]. Available: <https://www.visual-paradigm.com/guide/agile-software-development/user-story-vs-use-case/>

[94] А. В. Марков, “Совокупное использование сетей петри и uml-диаграмм при разработке программного обеспечения”, *Сборник научных трудов ИГТУ*, № 2, с. 85 – 94, 2011.

[95] G. Denaro, M. Pezze, “Petri Nets and Software Engineering”, in *Computer Science*, vol. 29, pp. 439 – 466, 2003.

[96] Th. Weng Jie, and M. Ariff Ameen, “A Model Driven method to represent Free Choice Petri Nets as Sequence Diagram”, *Software Engineering and Computer Systems*, №14, pp. 23 – 32, 2015.

[97] H. Singh, and Pr. Pal. Software Reliability Testing using Monte Carlo Methods International, *Journal of Computer Applications*, no. 4, pp. 41 – 44, 2013.

[98] Т. Зайко. Ассоциативные правила в интеллектуальном анализе данных, *Вестник Национального технического университета Харьковский политехнический институт*, № 39, с. 82 – 96, 2013.

[99] C. Zhang. *Association Rule Mining, Models and Algorithms*. Berlin, Germany: Springer, 2001.

[100] J. Adamo, *Data Mining for Association Rules and Sequential Patterns: Sequential and Parallel Algorithms*, NY, USA: Springer, 2001.

[101] Y. S. Koh, and N. Rountree, *Rare Association Rule Mining and Knowledge Discovery: Technologies for Infrequent and Critical Event Detection*. Hershey, PA: IGI Publishing, 2009.

[102] X. Piao, Z. Wang, and G.Liu. “Research on Mining Positive and Negative Association Rules Based on Dual Confidence” in *Fifth International Conference on Internet Computing for Science and Engineering*, Heilongjiang, 2010, pp. 102 – 105.

[103] Y. Ke, J. Cheng, and W. Ng, “An information-theoretic approach to quantitative association rule mining”, *Knowledge and Information Systems*, pp. 213–244, 2008.

[104] H. Nam, “Identification of temporal association rules from time-series microarray data sets”, *Proceedings of the 2nd international workshop on Data and text mining in bioinformatics*, California, 2008, pp. 21 – 28.

[105] R. Agrawal, and R. Srikant, “Fast Algorithms for Mining Association Rules in Large Databases”, *Proceedings of the 20th International Conference on Very Large Data Bases*, Chile, 1994, pp. 487 – 499.

[106] R. Agrawal, and R. Srikant, “Fast algorithms for mining association rules”, *Proceedings of the 20th International Conference on Very Large Data Bases*, San Francisco, pp. 487 – 499, 1994.

[107] J. H. Mojdeh, and R. Z. Osmar, “A Study on Interestingness Measures for Associative Classifiers”, in *Proceedings of the 2010 ACM Symposium on Applied Computing*, pp. 1039 – 1046, 2010.

[108] Sh. Shang, X. Dong, and J. Li, “Algorithms for mining negative association rules in multi-database”, *Computer Engineering and Applications*, vol. 45, no. 24, pp. 150 – 152, 2009.

[109] Ассоциативные правила. Сравнительный анализ инструментария, 2019. [Электронный ресурс]. Доступно: https://www.researchgate.net/publication/315629798_Associativnye_pravila_Sravnitelnyj_analiz_instrumentaria.

[110] R. Natarajan, and B. Shekar, “Interestingness of association rules in data mining: Issues relevant to e-commerce”, *Sadhana*, pp. 291 – 309, 2005.

[111] J. Manimaran, Analysing the quality of Association Rules by Computing an Interestingness Measures, *Indian Journal of Science and Technology*, vol.8, no.15, 2015.

[112] P. Fournier-Viger. How to auto-adjust the minimum support threshold according to the data size, Philippe Fournier-Viger. 2019. [Online]. Available: <http://data-mining.philippe-fournier-viger.com/how-to-auto-adjust-the-minimum-support-threshold-according-to-the-data-size/>.

[113] P. Fournier-Viger. “Un modèle hybride pour le support à l'apprentissage dans les domaines procéduraux et mal-définis”, Ph.D. Thesis, University of Quebec in Montreal, Montreal, Canada, 2010.

[114] J. Manimaran, and T. Velmurugan, “A survey of Association Rule Mining in Text Applications”, in *IEEE International Conference on Computational Intelligence and Computing Research*, pp. 698 – 702, 2013.

[115] R. Agrawal, and R. Srikant. “Mining sequential patterns”, *ICDE '95 Proceedings of the Eleventh International Conference on Data Engineering*, Washington, 1995, pp. 3 – 14.

[116] J. Han, J. Pei, and Y. Yin. “Mining frequent patterns without candidate generation”, *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, New York, 2010, pp. 1 – 12.

[117] R. Raval, J. Jeet Rajput, and V. Gupta. “Survey on several improved Apriori algorithms”, *IOSR Journal of Computer Engineering*, 2013, pp. 57 – 61.

[118] F. Sha, and Zh. Hangjun, “The Research and Improvement on the Apriori Algorithm of the Association Rule Mining”, *2nd International Workshop on Intelligent Systems and Applications*, Wuhan, 2013, pp. 110 – 114.

[119] J. Hipp. “Algorithms for Association Rule Mining – A General. Survey and Comparison”, *ACM SIGKDD Explorations Newsletter*, New York, 2000, pp. 58 – 64.

[120] D. Tsur, J. Ullman, S. Abiteboul, and Ch. Clifton, “Query flocks: a generalization of association-rule mining”, in *Proceedings of the 1998 ACM SIGMOD international conference on Management of data*, pp.1 – 12, 1998.

[121] A. Swami, R. Agrawal, and T. Imielinski, “Mining Association Rules between Sets of Items in Large Databases”, *International Conference on Management of Data*, Washington, 1993, pp. 207 – 216.

[122] M. Danubianu, St. Pentiuc, and I. Tobolcea, “Set-Oriented Mining for Association Rules in Relational Databases”, *The Sixth International Multi-Conference on Computing in the Global Information Technology*, Luxembourg, pp. 14 – 19, 2011.

[123] D. J. Lubinsky, “Discovery from Databases: A Review of AI and Statistical Techniques”, in *IJCAI-89 Workshop on Knowledge Discovery in Databases*, pp. 204 – 218, 1989.

[124] A. Trupti, and V. Santoch, “An Overview of Association Rule Mining Algorithms”, *International Journal of Computer Science and Information Technologies*, vol. 5, no. 1, pp. 927 – 930, 2014.

[125] A. Savasere, E. Omiecinski, and B. Navathe. “An Efficient Algorithm for Mining Association Rules in Large Databases”, *Proceedings of the 21th International Conference on Data Bases*, San Francisco, 1995, pp. 432 – 444.

[126] M. Zaki, and S. Parthasarathy, M. Ogihara, and Wei Li, “New Algorithms for Fast Discovery of Association Rules”, *KDD-97 Proceedings*, pp. 283 – 286, 1997.

[127] M. Klemettinen, H. Mannila, P. Ronkainen, H. Toivonen, and A. Inkeri Verkamo. “Finding interesting rules from large sets of discovered association rules”, in *Proceedings of the Third International Conference on Information and Knowledge Jlanagement*, pp. 401 – 407, 1994.

[128] G. Grahne, and J. Zhu, “Fast algorithms for frequent itemset mining using FP-trees”, in *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 10, pp. 1347 – 1362, 2005.

[129] S. Archana, and D. Elangovan, “Survey of classifications techniques in Data Mining”, *International Journal of Computer Science and Mobile Applications*, vol. 2, 2014, pp. 65 – 71.

[130] A. A. Freitas, *Data Mining and Knowledge Discovery with Evolutionary Algorithms*, Berlin, Germany: Springer-Verlag, 2002.

[131] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and regression trees*, USA: Chapman & Hall, 1984.

[132] CSE5230 Tutorial: The ID3 Decision Tree Algorithm, Monash university. 2019. [Online]. Available: http://web.info.uvt.ro/~dzaharie/dm2017/EN/projects/Algorithms/DecisionTrees/DecisionTrees_ID3Tutorial.pdf.

[133] ID3 ALGORITHM. 2019. [Online]. Available: <https://www.slideshare.net/HARDIKSINGH7/id3-algorithm-56632554>

[134] G. A. Wang, “Survey on Training Algorithms for Support Vector Machine Classifiers”, *Fourth International Conference on Networked Computing and Advanced Information Management*, Gyeongju, 2008.

[135] L. Auria, and R. A. Moro, “Vector Machines (SVM) as a Technique for Solvency Analysis”, *SSRN Electronic Journal*, vol. 1, no. 1, pp. 1 – 8, 2008.

[136] Классификатор kNN. 2019. [Электронный ресурс]. Доступно: <https://habrahabr.ru/post/149693/>.

[137] K-nearest neighbours. 2019. [Online]. Available: <http://www.cs.upc.edu/~bejar/apren/docum/trans/03d-algind-knn-eng.pdf>.

[138] P. Bhargavi, and Dr. S. Jyothi Soil, Classification Using Data Mining Techniques: A Comparative Study, in *International Journal of Engineering Trends and Technology*, 2011.

[139] С. В. Субботин, и Д. Ю. Большаков, “Применение байесовского классификатора для распознавания классов целей”, *Журнал радиоэлектроники*, №4, 2006.

[140] Naive Bayes Classifier 2019. [Online]. Available: <http://responsive.media.mit.edu/wp-content/uploads/sites/5/2014/01/Class-4-Naive-Bayes.pdf>

[141] C. Maffort. Mining Architectural Patterns Using Association Rules. *International Conference on Software Engineering and Knowledge Engineering*, Boston, 2013.

[142] M. Azzeh. Software Stage-Effort Estimation Using Association Rule Mining and Fuzzy Set Theory. 2019. [Online]. Available: <https://arxiv.org/ftp/arxiv/papers/1703/1703.04539.pdf>.

[143] V. Dubovoy, I. Pylypenko, and R. Stets, *Markov model application for analyzing cyclicity influence on the branched technological process management*, SWVNTU, no. 4, 2015.

[144] В. И. Тихонов, и М. А. Миронов, *Марковские процессы*, Москва, Россия : Советское радио, 1977.

[145] Великий тлумачний словник сучасної української мови: 250000 / уклад. та голов. ред. В. Т. Бусел, Київ, Україна: Перун, 2005.

[146] Е. Б. Дынкин, Марковские процессы и связанные с ними задачи анализа, *УМН*, № 2, 1960.

[147] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, *Pattern Oriented Software Architecture*, New Jersey, USA, 2002.

[148] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, Boston, USA: Wesley, 2003.

- [149] Э. Дж. Брауде, *Технология разработки программного обеспечения*, Санкт-Петербург, Россия: Питер, 2004.
- [150] MVC xerox парс. 2019. [Online]. Available: <https://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html>
- [151] С. Рогачев, *Обобщенный Model-View-Controller. Каркас на основе шаблона проектирования MVC*, 2019. [Электронный ресурс]. Доступно: <https://rsdn.org/article/patterns/generic-mvc.xml>.
- [152] MVC Patterns 2019. [Online]. Available: <http://aalmiray.github.io/griffon-patterns/>
- [153] Д. Чедвик, Т. Снайдер, и Х. Панда. *ASP.NET MVC 4. Разработка реальных веб-приложений с помощью ASP.NET MVC*. Farnham, Surrey: O'reilly, 2013.
- [154] J. Rumbaugh, I. Jacobson, and G. Booch Addison. *The unified modeling language reference manual*. Boston, USA: Addison-Wesley, 1999.
- [155] О. М. Бевз, В. М. Папінов, та Ю. А. Скидан. *Проектування програмних засобів систем управління*. Вінниця, Україна: ВНТУ, 2010.
- [156] Е. Гемма, Р. Хелм, *Приемы объектно-ориентированного проектирования. Паттерны проектирования*, СПб, Россия: Питер, 2001.
- [157] Отношения классов – от UML к коду, 2019. [Электронный ресурс]. Доступно: <https://habr.com/ru/post/150041/>.
- [158] SQL или NoSQL – вот в чём вопрос. 2019. [Электронный ресурс]. Доступно: <https://habr.com/company/ruvds/blog/324936/>.
- [159] О. Н. Романюк, та Т. О. Савчук, *Організація баз даних і знань: навчальний посібник*. Вінниця, Україна: УНІВЕРСУМ-Вінниця, 2003.
- [160] J. MacLennan, Z. Tang, and B. Crivat, *Data Mining with Microsoft SQL Server 2005*. Indianapolis, USA: Wiley Publishing, 2008.
- [161] Д. Боуман, С. Эмерсон, и М. Дарновски, *Практическое руководство по SQL*. New-York, USA: Williams, 2002.

ДОДАТКИ

ДОДАТОК А

АКТИ ВПРОВАДЖЕННЯ РЕЗУЛЬТАТІВ ДИСЕРТАЦІЇ

ЗАТВЕРДЖУЮ
Директор ТОВ «Бізнес-Програми»
С. М. Бондаренко
07 2018 р.



АКТ

впровадження результатів дисертаційної роботи
Приймак Наталії Василівни

«ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ ПОШУКУ АСОЦІАТИВНИХ ПРАВИЛ ЩОДО ТРИВАЛОСТІ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ»

Комісія у складі начальника відділу розробки програмного забезпечення Созонтова О. В. та консультанта з програмного забезпечення Перит І.О. встановила, що результати дисертаційної роботи Приймак Наталії Василівни використані при розробці програмного забезпечення ТОВ «Бізнес-Програми» для визначення тривалості виконання завдання програмістом певної кваліфікації. Використання знайдених закономірностей щодо тривалості розробки програмного забезпечення, а також застосування запропонованої автором відповідної інформаційної технології дозволило покращити процес розробки програмного забезпечення.

Комісія підтверджує, що використані такі результати дисертаційної роботи:

- метод визначення частих предметних наборів для пошуку асоціативних правил щодо тривалості розробки програмного забезпечення;
- алгоритм використання знайдених асоціативних правил щодо тривалості розробки програмного забезпечення;
- програмні засоби пошуку та використання асоціативних правил щодо тривалості розробки програмного забезпечення.

Ефект від впровадження зазначених результатів полягає у підвищенні якості процесу розробки програмного забезпечення, а саме у: зростанні точності прогнозування тривалості виконання завдання розробником певної кваліфікації в 1,7 рази; скороченні тривалості визначення часу, необхідного на виконання завдання розробником до 6 разів.

Члени комісії:

Начальник відділу розробки

програмного забезпечення

Консультант з програмного забезпечення



О.В. Созонтов

І.О. Перит

УЗГОДЖЕНО

Проректор з наукової роботи

В. Павлов
2018 р.

ЗАТВЕРДЖУЮ

Директор ТОВ «Бізнес-Програми»

С.М. Бондаренко
09 2018 р.

АКТ УПРОВАДЖЕННЯ № Результатів науково-дослідної роботи

Цим актом підтверджується, що результати науково-дослідницької роботи
«Розробка інформаційної системи пошуку асоціативних правил щодо
тривалості розробки програмного забезпечення»
 (Найменування НДР)

що виконана Вінницьким національним технічним університетом
 за договором про творчу співпрацю № 47/5 від 15 січня 2018 р.
 (господарською або держбюджетною темою, за договором про творчу співдружність, в межах робочого часу викладача)

упроваджено в 2018 році
 строк упровадження (рік)

у ТОВ «Бізнес-Програми»
 (найменування підприємства, організації, установи тощо, де здійснюється упровадження)

1. Вид упровадженої продукції інформаційна система пошуку асоціативних
правил щодо тривалості розробки програмного забезпечення
 (техніка, технологія, матеріал, методи, теорії і інше)
2. Характеристика масштабу упровадження одиничне
 (одиничне, дрібносерійне, серійне, інше)
3. Новизна результатів науково-дослідної роботи
принципово нові
 (піонерські, принципово нові, якісно нові, модифікації, модернізація старих розробок, інше)
4. Річний економічний ефект буде визначений по завершенню дослідної
експлуатації програмного забезпечення
5. Соціальний та науково-технічний ефект
Оптимізація процесу визначення часу необхідного для виконання завдання, що
дозволило автоматизувати процес планування розробки програмного забезпечення
 (охорона навколишнього середовища, поліпшення й оздоровлення умов праці, удосконалення структури керування, розвиток нових технологій в освіті, інше)

Від ВНТУ

Начальник ІДЧ

В. В. Богачук
 Науковий керівник
Т. О. Савчук

Від ТОВ «Бізнес-Програми»

Директор



С.М. Бондаренко

ДОДАТОК Б

СПИСОК ПУБЛІКАЦІЙ ЗА ТЕМОЮ ДИСЕРТАЦІЇ

[1] Т. О. Савчук, та Н. В. Приймак, “Обґрунтування доцільності використання методу *fp-growth* (*fpg*) при розробці програмного забезпечення”, *Вчені записки ТНУ імені В.І. Вернадського. Серія Технічні науки*, № 5, с. 41 – 45, 2018.

[2] Т. О. Савчук, та Н. В. Приймак, “Розробка інформаційної моделі процесу пошуку асоціативних правил при розробці програмного забезпечення”, *Інформаційні технології та комп’ютерна інженерія*, vol 42, № 2, с. 43 – 48, 2018.

[3] Т. О. Савчук, та Н. В. Приймак, “Розробка інформаційної технології процесу пошуку асоціативних правил при розробці програмного забезпечення”, *Наукові праці : наук. журн. / Чорном. нац. ун-т ім. П. Могили. Комп’ютерні технології*, вип. 305, т. 317, с. 6 – 11, 2018.

[4] T. O. Savchuk, and N. V. Pryimak, “Modeling of software development process with the Markov processes”, *Eastern-European jornal of Enterprise Technologies . Information technology*, vol. 3, no. 2, pp. 33 – 38, 2017.

[5] T. O. Savchuk, N. V. Pryimak, A. Assembay, T. Zyska, M. Junisbekov, and A. Annabaev “The technology of searching the associative rules while developing the software”, *Proc. SPIE 10445, Photonics Applications in Astronomy, Communications, Industry, and High Energy Physics Experiments*, 2017, doi: 10.1117/12.2280900.

[6] Т. О. Савчук, Н. В. Приймак, “Використання методів інтелектуального аналізу даних при тестуванні програмного забезпечення”, на *X міжнародній конференції «Інтернет-освіта-наука-2016»*, Вінниця, 2016, с. 28 – 29.

[7] Т. О. Савчук, та Н. В. Приймак, “Використання методів штучного інтелекту для прогнозування часу для виправлення дефектів розробки програмного забезпечення”, на *X міжнародній конференції «Інтернет-освіта-наука-2016»*, Вінниця, 2016, с. 30 – 31.

[8] Т. О. Савчук, та Н. В. Приймак, “Обґрунтування вибору методу визначення частих предметних наборів для пошуку асоціативних правил при розробці програмного забезпечення”, на *XI міжнародній конференції «Інтернет-освіта-наука-2018»*, Вінниця, 2018, с. 45 – 46.

[9] Т. О. Савчук, та Н. В. Приймак, “Інформаційна модель процесу пошук і використання асоціативних правил при розробці програмного забезпечення”, на *XI міжнародній конференції «Інтернет-освіта-наука-2018»*, Вінниця, 2018, с. 43 – 44.

[10] Т. О. Savchuk, and N. V. Pryimak, “Feasibility study of using associative rules during the software development”, on *IX international Conference of the Georgian Mathematical Union*, Batumi, 2018, pp. 203 – 204.

[11] Т. О. Савчук, та Н. В. Приймак, “Інформаційна технологія пошуку асоціативних правил при розробці програмного забезпечення”, на *Всеукраїнській наук.-практ. конф. здобувачів вищої освіти й молодих учених*, Кропивницький, 2018, с. 100 – 102.

[12] Т. О. Савчук, та Н. В. Приймак, “Використання frg-алгоритму для пошуку асоціативних правил при прийнятті рішень в управлінні процесами”, на *XLV НТК проф.-викл. складу, співробітників та студентів університету*, Вінниця, 2016, с. 929 – 930.

[13] Т. О. Савчук, та Н. В. Приймак, “Використання Марковських ланцюгів при моделюванні процесу розробки програмного забезпечення”, на *XLVI НТК проф.-викл. складу, співробітників та студентів університету*, Вінниця, 2017, с. 1019 – 1020.

[14] Т. О. Савчук, та Н. В. Приймак, “Метод пошуку асоціативних правил при розробці програмного забезпечення, що базується на попередньому розбитті множини завдань” на *XLVIII НТК проф.-викл. складу, співробітників та студентів університету*, Вінниця, 2019, с. 655 – 656.

[15] Т. О. Савчук, Н. В. Приймак, “Інформаційна система пошуку асоціативних правил при розробці програмного забезпечення”, *Свідцтво про реєстрацію авторського права на твір №82173*; Жов. 11, 2018.

ДОДАТОК В

ЙМОВІРНОСТІ ПЕРЕХОДІВ МІЖ СТАНАМИ ДЛЯ ТРЬОХ РІВНІВ

ДЕТАЛІЗАЦІЇ ПРОЦЕСУ РОЗРОБКИ ПРОГРАМНОГО

ЗАБЕЗПЕЧЕННЯ

Рівень деталізації розробки ПЗ		Ймовірності переходів між станами															
		P_{11}	P_{12}	P_{13}	P_{14}	P_{21}	P_{22}	P_{23}	P_{24}	P_{31}	P_{32}	P_{33}	P_{34}	P_{41}	P_{42}	P_{43}	P_{44}
ПРОЕКТ 1	I рів.	0.1	0.9	0	-	0	0.1	0.9	-	0.9	0	0.1	-	-	-	-	-
	II рів.	0.2	0.8	0	0	0	0.2	0.8	0	0	0	0.2	0.8	0.8	0	0	0.2
	III рів.	0.3	0.7	0	-	0	0.3	0.7	-	0.7	0	0.3	-	-	-	-	-
ПРОЕКТ 2	I рів.	0.2	0.8	0	-	0	0.2	0.8	-	0.8	0	0.2	-	-	-	-	-
	II рів.	0.25	0.75	0	0	0	0.25	0.75	0	0	0	0.25	0.75	0.75	0	0	0.25
	III рів.	0.15	0.85	0	-	0	0.15	0.85	-	0.85	0	0.15	-	-	-	-	-
ПРОЕКТ 3	I рів.	0.15	0.85	0	-	0	0.15	0.85	-	0.85	0	0.15	-	-	-	-	-
	II рів.	0.3	0.7	0	0	0	0.3	0.7	0	0	0	0.3	0.7	0.7	0	0	0.3
	III рів.	0.1	0.9	0	-	0	0.1	0.9	-	0.1	0	0.9	-	-	-	-	-
ПРОЕКТ 4	I рів.	0.35	0.65	0	-	0	0.35	0.65	-	0.65	0	0.35	-	-	-	-	-
	II рів.	0.15	0.85	0	0	0	0.15	0.85	0	0	0	0.15	0.85	0.85	0	0	0.15
	III рів.	0.25	0.75	0	-	0	0.25	0.75	-	0.75	0	0.25	-	-	-	-	-
ПРОЕКТ 5	I рів.	0.3	0.7	0	-	0	0.3	0.7	-	0.7	0	0.3	-	-	-	-	-
	II рів.	0.2	0.8	0	0	0	0.2	0.8	0	0	0	0.2	0.8	0.8	0	0	0.2
	III рів.	0.3	0.7	0	-	0	0.3	0.7	-	0.7	0	0.3	-	-	-	-	-

ДОДАТОК Г

ЛІСТИНГ ПРОГРАМНОЇ РЕАЛІЗАЦІЇ ОСНОВНИХ МОДУЛІВ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ПОШУКУ АСОЦІАТИВНИХ ПРАВИЛ ЩОДО ТРИВАЛОСТІ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Модуль класифікації даних

```
public class C45 implements Classifier {

    public Tree decisionTree;
    private Results results;

    @Override
    public void train(final PreprocessedData ppd) {
        final HashMap attributes = ppd.getAttributes();
        // choose target - assuming that target is a last column
        final Attribute targetAttribute = (Attribute) attributes.get(ppd.getTargetName());
        final List<CSVRecord> dataSet = ppd.getTrainingDataSet();
        TreeConstructor treeConstructor = new TreeConstructor(dataSet, attributes, targetAttribute);
        this.decisionTree = treeConstructor.getDecisionTree();
    }

    @Override
    public void test(final PreprocessedData ppd) {

        //Get the testing subset of examples
        final List<CSVRecord> dataSet = ppd.getTestingDataSet();

        try {

            //Create a tempfile to store records, if it exists already delete it
            String tempFilePath = System.currentTimeMillis() + "_Results.csv";

            if (new File(tempFilePath).isFile()) {
                new File(tempFilePath).delete();
            }
            File file = File.createTempFile(System.currentTimeMillis() + "_Results", ".csv");

            //Print the header
            PrintWriter printWriter = new PrintWriter(new FileOutputStream(file, false));
            printWriter.println("actual,predicted,error");

            //For each record, store the actual target value.
            for (CSVRecord instance : dataSet) {
```

```

//For each record, store the actual target value.
String actual = instance.get(ppd.getTargetName());

//Classify the instance using the classifier and store the predicted target value
String predicted = classify(instance);

//If the two values don't match, mark the error column
String error = actual.equals(predicted) ? "" : "+";

    printWriter.println(actual + "," + predicted + "," + error);
}

printWriter.close();

try {
    //Create a new results instance using this tempfile
    results = new Results(file);
} catch (Exception e) {
    e.printStackTrace();
}

} catch (IOException e) {
    e.printStackTrace();
}
}

@Override
public String classify(final CSVRecord instance) {
    return decisionTree.search(decisionTree, instance);
}

@Override
public Results getResults() {
    return results;
}

public static void main(String[] args) throws Exception {
    String training = Converter.toCsv(new ExcelReader()
        .table(new File("src/main/resources/final_marked.xls")), "train");
    String testing = Converter.toCsv(new ExcelReader()
        .table(new File("src/main/resources/final.xls")), "test");

    PreprocessedData trainPpd = new PreprocessedData(training);
    trainPpd.splitTrainingTestPercentage(1);
    PreprocessedData testPpd = new PreprocessedData(training);
    PreprocessedData classify = new PreprocessedData(testing);
    testPpd.splitTrainingTestPercentage(0);

    C45 c45 = new C45();
    c45.train(trainPpd);
    c45.test(testPpd);

```

```

        c45.getResults().records.forEach(System.out::println);

        System.out.println(c45.classify((CSVRecord) classify.getDataSet().get(classify.getDataSet().size() - 8)));
    }
}

public interface Classifier {
    void train(PreprocessedData ppd);

    void test(PreprocessedData ppd);

    String classify(CSVRecord instance);

    Results getResults();
}

```

Модуль побудови дерева

```

public class Tree implements Serializable {

    private Attribute attribute;
    private String nodeName;
    private String value;
    private final HashMap<String, Tree> childrenDiscrete = Maps.newLinkedHashMap();
    private final List<Tree> childrenContinuous = Lists.newLinkedList();

    public Tree(final String label, Attribute targetAttribute) {
        this.nodeName = label;
        this.value = nodeName;
        this.attribute = targetAttribute;
    }

    public Tree(final Gain gain) {
        this.nodeName = gain.getAttributeName();
        this.value = gain.getValue();
        this.attribute = gain.getAttribute();
    }

    public void addChild(final Tree child) {
        final ArrayList children = new ArrayList<>(this.childrenContinuous);
        if (child.attribute.isTarget() && children.size() > 0) {
            if (child.nodeName.equals(children.get(0))) {
                this.nodeName = child.nodeName;
                this.attribute = child.attribute;
                return;
            }
        }
        this.childrenContinuous.add(child);
    }

    public void addChild(Attribute attribute, final String childName, final Tree child) {

```

```

        this.attribute = attribute;
        this.childrenDiscrete.put(childName, child);
    }

    public String search(final Tree node, final CSVRecord instance) {

        if (node.attribute.isTarget()) {
            return node.nodeName;
        }

        final String value = instance.get(node.nodeName);
        if (node.attribute.isContinuous()) {
            // check if value smaller or greater than threshold and continue search recursively
            int direction = new BigDecimal(value).compareTo(new BigDecimal(node.value)) <= 0 ?
                0 : // left
                1; // right
            return search(node.childrenContinuous.get(direction), instance);
        }
        try {
            return search(node.childrenDiscrete.get(value), instance);
        } catch (Exception e) {
            return " ";
        }
    }
}

public class TreeConstructor {

    Tree root;

    public TreeConstructor(final List<CSVRecord> dataSet, final HashMap attributes, final Attribute targetAttribute)
    {
        this.root = constructDecisionTree(dataSet, attributes, targetAttribute);
    }

    private static Tree constructDecisionTree(final List<CSVRecord> dataSet, final HashMap<String, Attribute>
attributes, final Attribute target) {

        // Calculate Target Entropy
        final double targetEntropy = calculateEntropy(dataSet, target);
        // if entropy = 0 create leaf
        if (!(targetEntropy > 0)) {
            // extract label
            final CSVRecord instance = Iterables.get(dataSet, 0);

            System.out.println("\n-----" + instance.get(target.getName()) + "\n\n");
            // create and return new node of target type
            return new Tree(instance.get(target.getName()), target);
        }

        // holds information on the attribute that gives the maximum gain
        final Gain maximumGain = getMaximumGain(dataSet, attributes, target, targetEntropy);

```

```

dataSet.removeAll(dataSet); // remove processed data

if (Pruning.getSplitCriterion(maximumGain))
    System.out.println("\nFeature: " + maximumGain.getAttributeName() + "\nGAIN: " +
maximumGain.getGain() + "\nTHRESHOLD: " + maximumGain.getValue() + "\nSPLIT : " +
Pruning.getSplitCriterion(maximumGain) + "\n");
public static Gain getContinuousAttributeGain(final List<CSVRecord> dataSet, final Attribute attribute, final
Attribute target, final double targetEntropy, final String threshold) {
    // holds subsets of data, below and above the threshold
    final List instancesBelowThreshold = Lists.newArrayList();
    final List instancesAboveThreshold = Lists.newArrayList();
    // for each instance in the dataSet
    dataSet.forEach(instance -> {
        // Get Attribute's value from the current instance
        final String value = instance.get(attribute.getName());
        // if value smaller equal than threshold, associate instance with the left subset, otherwise right
        if (new BigDecimal(value).compareTo(new BigDecimal(threshold)) <= 0) {
            instancesBelowThreshold.add(instance);
        } else {
            instancesAboveThreshold.add(instance);
        }
    });

    final double numInstancesBelowThreshold = instancesBelowThreshold.size();
    final double numInstancesAboveThreshold = instancesAboveThreshold.size();
    final double numInstances = numInstancesBelowThreshold + numInstancesAboveThreshold;
    final double probA = numInstancesBelowThreshold / numInstances;
    final double probB = numInstancesAboveThreshold / numInstances;
    // Calculate subset's entropy
    final double entropyA = calculateEntropy(instancesBelowThreshold, target);
    final double entropyB = calculateEntropy(instancesAboveThreshold, target);

    //Calculate gain for the given threshold
    final double gain = targetEntropy - (probA * entropyA) - (probB * entropyB)

    final HashMap decisionClassesLeft = countDecisionClassLabels(instancesBelowThreshold, target);
    final HashMap decisionClassesRight = countDecisionClassLabels(instancesAboveThreshold, target);

    return new Gain(attribute, entropyA, entropyB, threshold, gain, decisionClassesLeft, decisionClassesRight,
instancesBelowThreshold, instancesAboveThreshold, targetEntropy);
}
public static ArrayList createThresholdsForAttribute(final List<CSVRecord> dataSet, final Attribute attribute, final
Attribute target) {
    final TreeSet thresholds = new TreeSet<>();
    if (!attribute.isContinuous()) {

        dataSet.forEach(instance -> thresholds.add(instance.get(attribute.getName())));
        return new ArrayList(thresholds);
    }

    Collections.sort(dataSet, (r1, r2) -> r1.get(attribute.getName()).compareTo(r2.get(attribute.getName())));
    // extract thresholds by checking at what points the decision class changes

```

```

final CSVRecord[] previousInstance = {dataSet.get(0)};    dataSet.forEach(instance -> {

    final String previousLabel = previousInstance[0].get(target.getName());
    final String currentLabel = instance.get(target.getName());
        if (!currentLabel.equals(previousLabel)) {
            final String threshold = previousInstance[0].get(attribute.getName()).toString();
            thresholds.add(threshold);
        }
    previousInstance[0] = instance;

});
// return sorted list of unique thresholds
return new ArrayList(thresholds);

}

public static Double calculateEntropy(final List<CSVRecord> dataSet, final Attribute target) {
    final double numberOfSamples = dataSet.size();
    final HashMap countMap = countDecisionClassLabels(dataSet, target);
    final double[] entropy = {0};

    countMap.keySet().forEach(key -> {
        final double probabilityOfValue = (
            Double.parseDouble(
                String.valueOf(countMap.get(key)))
                /
                numberOfSamples);
        entropy[0] -= (probabilityOfValue * (Math.log(probabilityOfValue) / Math.log(2)));

    });
    return entropy[0];
}

public static HashMap countDecisionClassLabels(final List<CSVRecord> dataSet, final Attribute target) {
    final HashMap countMap = Maps.newHashMap();
    dataSet.forEach(instance -> {

        final String label = instance.get(target.getName()).toString();
        if (!countMap.containsKey(label)) {
            countMap.put(label, (double) 1);
        } else {
            Double count = (Double) countMap.get(label);
            count = count + 1;
            countMap.put(label, count);
        }
    });
    return countMap;
}

public Tree getDecisionTree() {
    return root;
}
}

```

```

public class FpgDriver {

    private static final FpgDriver driver = new FpgDriver();

    public static FpgDriver driver(){
        return driver;
    }

    private SparkConf conf = new SparkConf()
        .setAppName("Tasker")
        .setMaster("local");
    private JavaSparkContext sc = new JavaSparkContext(conf);
    private double minSupp = 0.1;
    public void setMinSupp(double minSupp) {
        this.minSupp = minSupp;
    }
    public double getMinSupp() {
        return minSupp;
    }

    private FpgDriver() {
    }

    public List<Rule> produceRules(List<List<String>> table) {

        JavaRDD<List<String>> transactions = sc.parallelize(table);

        FPGrowth fpg = new FPGrowth()
            .setMinSupport(minSupp)
            .setNumPartitions(6);
        FPGrowthModel<String> model = fpg.run(transactions);

        for (FPGrowth.FreqItemset<String> itemset: model.freqItemsets().toJavaRDD().collect()) {
            System.out.println("[ " + itemset.javaItems() + " ], " + itemset.freq());
        }
        double minConfidence = 0.5;
        return model.generateAssociationRules(minConfidence)
            .toJavaRDD().collect().stream()
            .map(rule -> new Rule(
                rule.javaAntecedent().toString(),
                rule.javaConsequent().toString(),
                rule.confidence()
            ))
            .peek(System.out::println)
            .collect(Collectors.toList());

    }
}

```

ДОДАТОК Д

ІНСТРУКЦІЯ КОРИСТУВАЧА ІНФОРМАЦІЙНОЇ СИСТЕМИ ПОШУКУ АСОЦІАТИВНИХ ПРАВИЛ ЩОДО ТРИВАЛОСТІ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Після запуску інформаційної системи пошуку асоціативних правил щодо тривалості розробки програмного забезпечення користувачу відкривається початкова сторінка даного ПЗ, якою є вікно автентифікації, відображене на рис. Д.1.

Для автентифікації та для логіна у систему користувач повинен заповнити три обов'язкові поля, щоб мати доступ лише до даних своєї компанії:

- логін;
- пароль,
- організацію, тобто назву компанії.

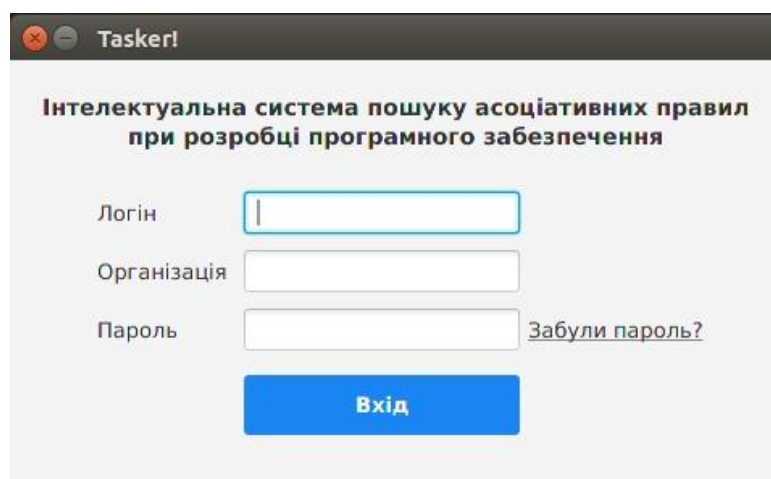


Рисунок Д.1 – Вікно автентифікації до ПЗ пошуку асоціативних правил щодо тривалості розробки програмного забезпечення

Якщо користувач забув пароль для входу в систему, він може скористуватися опцією «Забули пароль?» та отримати новий пароль на електронну адресу, що була використана при реєстрації.

Після логіну користувачу відображається головне вікно інформаційної системи, в якому зліва знаходиться меню з основними функціональними можливостями, а справа відображаються результати роботи ПЗ пошуку асоціативних правил щодо тривалості розробки програмного забезпечення (рис. Д.2).

Основними функціональними можливостями розробленого ПЗ є:

- пошук асоціативних правил серед завдань, виконаних під час розробки ПЗ;
- класифікація даних за допомогою алгоритму C4.5;
- аналіз завдань, з метою визначення часу, необхідного на їхнє виконання розробником.

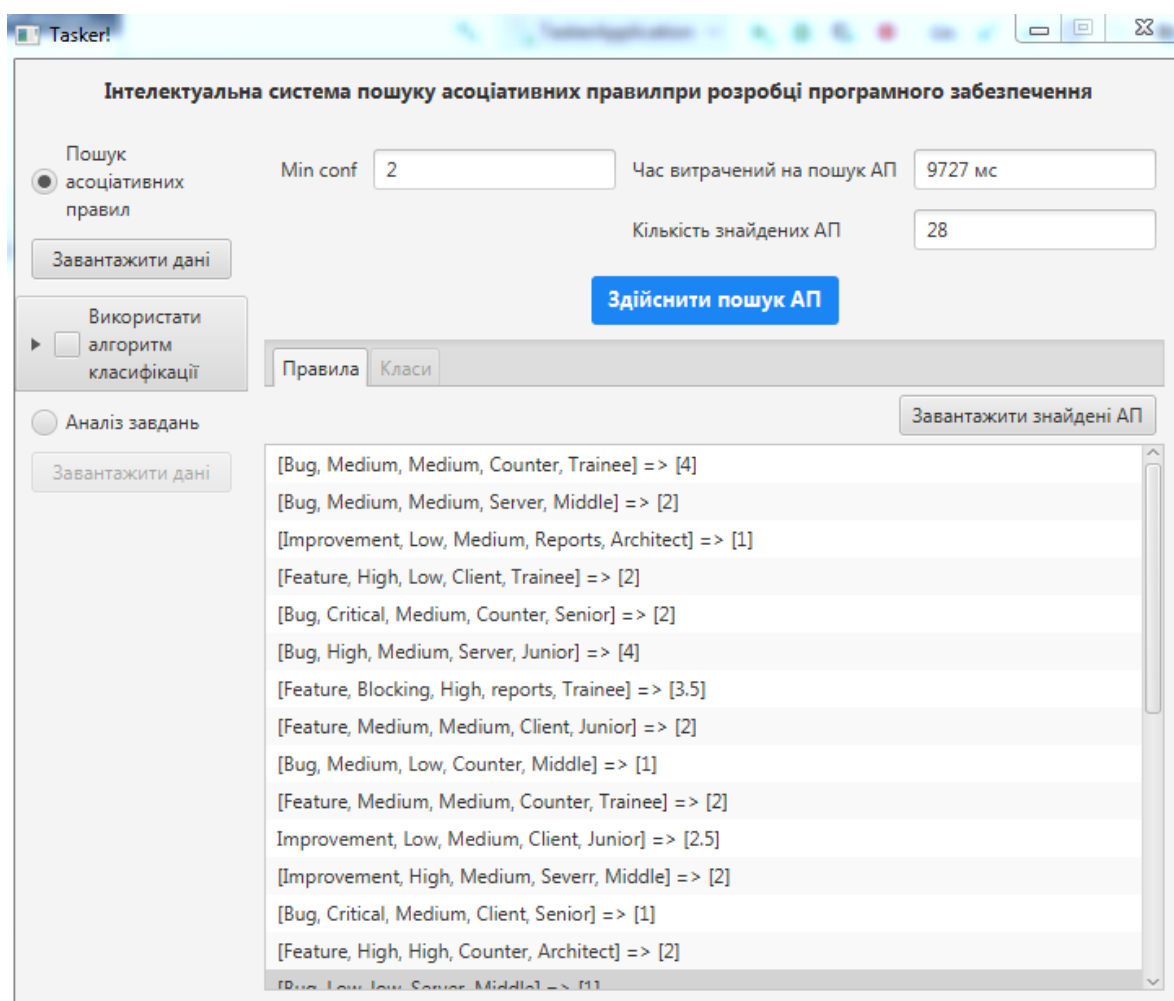


Рисунок Д.2 – Головне вікно програмного забезпечення пошуку асоціативних правил щодо тривалості розробки ПЗ

Якщо АП для цієї компанії вже були знайдені раніше, то після автентифікації користувачу вже не потрібно навчати алгоритм класифікації чи здійснювати пошук АП. Він одразу може використовувати попередньо знайдені АП для визначення часу, необхідного на реалізацію певного завдання розробником відповідної кваліфікації, обравши відповідну опцію «Аналіз завдань» (рис. Д.3).

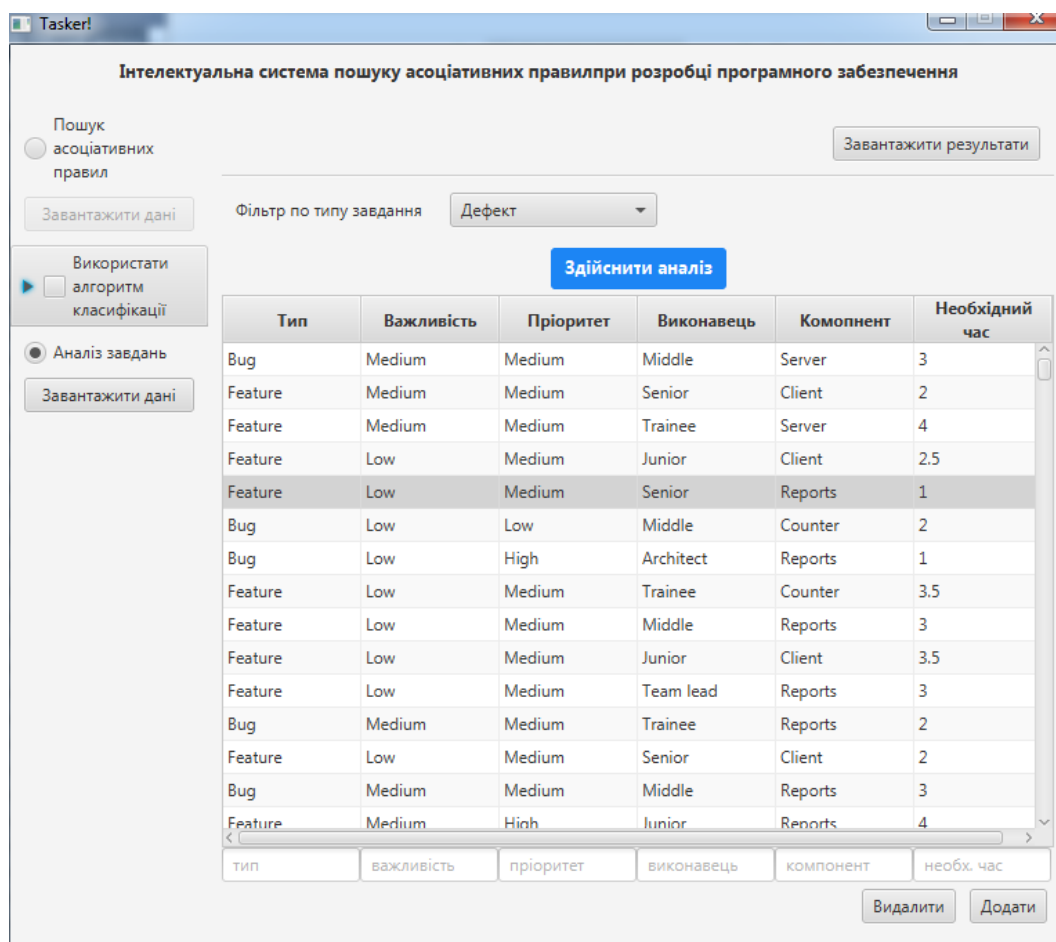


Рисунок Д.3 – Вікно аналізу завдань програмного забезпечення пошуку асоціативних правил щодо тривалості розробки ПЗ

При цьому користувачу відкривається нове вікно, де він може переглянути усі завдання з часом, необхідним на їх реалізацію розробником. Для визначення часу користувач повинен завантажити завдання, натиснувши кнопку «Завантажити дані», після чого завдання із визначеним часом будуть відображатися у таблиці.

Оскільки завдання мають різний тип (дефект, особливість, задача), то їх можна фільтрувати відповідно до типу, для зручності їх використання та аналізу. Також результати даної функціональності можна завантажити у вигляді xls файлу, натиснувши кнопку «Завантажити результати».

Якщо пошук АП для цієї компанії здійснюється вперше, то користувач може обрати два способи їх пошуку (опція «Використати алгоритм класифікації»):

- з попередньою класифікацією даних;
- без попередньої класифікації даних.

Для пошуку асоціативних правил без попередньої класифікації даних, користувачу потрібно обрати опцію «Пошук АП» (рис. Д.2) та ввести значення *minconf*, на основі якого буде обраховуватися значення *minsupp*. Натиснути кнопку «Завантажити дані» та завантажити завдання, серед яких буде здійснюватися пошук асоціативних правил. Про успішне завантаження даних користувача буде повідомлено за допомогою відповідного вікна (рис. Д.4):

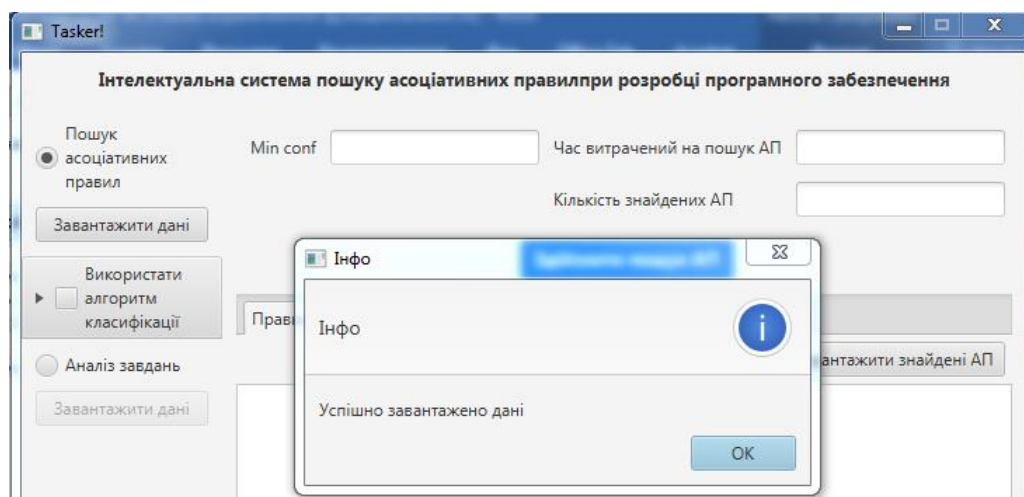


Рисунок Д.4 – Повідомлення про успішне завантаження даних

Далі необхідно натиснути кнопку «Здійснити пошук АП» та очікувати на відображення відповідного інформаційного повідомлення про завершення пошуку АП (рис. Д.5) та результатів у вкладці «Правила».

У полях «Час витрачений на пошук АП» та «Кількість знайдених АП» відображається тривалість пошуку АП та їх кількість відповідно. Для зручності використання знайдених АП їх можна завантажити, використавши кнопку «Завантажити знайдені АП».

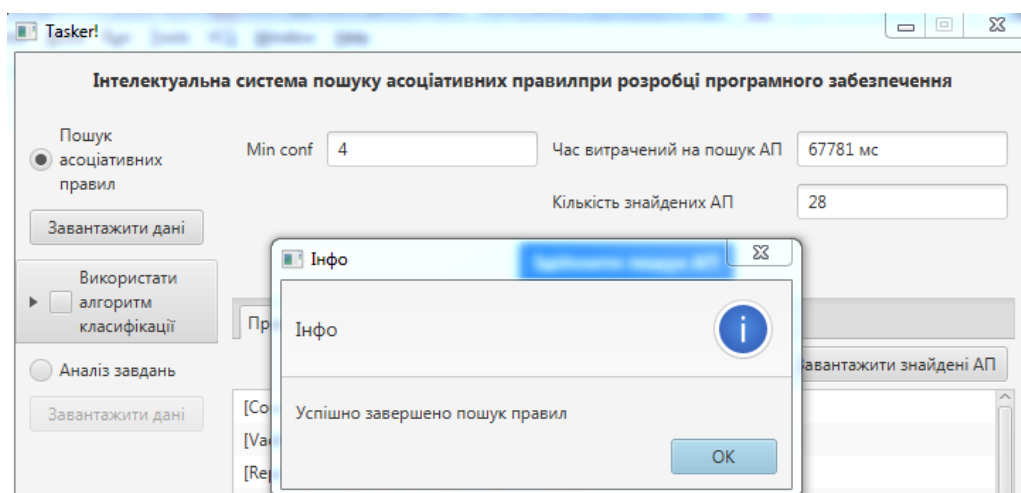


Рисунок Д.5 – Повідомлення про успішне завершення пошуку АП

Якщо користувач обирає варіант пошуку асоціативних правил з попередньою класифікацією завдань, то спочатку потрібно навчити алгоритм класифікації C4.5 (рис. Д.2).

Для цього потрібно обрати опцію «Використати алгоритм класифікації»; завантажити дані для тренування алгоритму, натиснувши кнопку «Навчальна вибірка»; запустити процес навчання, натиснувши кнопку «Навчати» та слідкувати за прогресом навчання.

Якщо вищезгадана опція вибрана, то класифікація завдань буде здійснена автоматично перед початком пошуку АП. Для класифікації завдань, серед яких необхідно знайти асоціативні правила, спочатку необхідно їх завантажити, використавши кнопку «Завантажити дані». Після цього класифіковані на три класи завдання відображатимуться в окремих блоках у вкладці «Класи» (рис. Д.6). Для зручності аналізу класифікованих завдань, їх можна завантажити, використавши кнопку «Завантажити класи».

Рисунок Д.6 – Завдання класифіковані програмним забезпеченням пошуку АП щодо тривалості розробки ПЗ

Для пошуку АП серед класифікованих завдань потрібно ввести minconf та натиснути кнопку «Здійснити пошук АП», оскільки процедура аналогічна до пошуку АП без попередньої класифікації.

Завдання (як для класифікації так і для пошуку АП) потрібно завантажити у вигляді xsl файлу заданого формату (рис. Д.7).

1	Issue Type	Severity	Priority	Assignee	Component/s	Time Spent
2	Bug	Medium	Medium	middle	Server	3
3	Feature	Medium	Medium	junior	Server	3
4	Feature	Medium	Medium	middle	Server	3
5	Feature	Low	Medium	trainee	Server	3
6	Feature	Low	Medium	junior	Reports	3
7	Bug	Low	Low	middle	Reports	3
8	Bug	Low	High	junior	Reports	3
9	Feature	Low	Medium	middle	Reports	3
10	Feature	Low	Medium	middle	Reports	3
11	Feature	Low	Medium	middle	Reports	3
12	Feature	Low	Medium	middle	Reports	3
13	Bug	Medium	Medium	middle	Reports	1

Рисунок Д.7 – Зразок оформлення xsl файлу із завданнями, серед яких потрібно здійснювати пошук асоціативних правил

ДОДАТОК Е

ВИБІРКА ЗАВДАНЬ ДЛЯ ПОШУКУ АСОЦІАТИВНИХ ПРАВИЛ

№	Тривалість	Рівень кваліфікації	Компонент ПЗ	Критичність	Складність	Пріоритет	Тип	Стать розробника	Реліз ПЗ
1	2	middle	Server	blocking	medium	High	дефект	men	реліз 1
2	3,5	junior	Archive	medium	medium	High	особливість	women	реліз 1
3	1,5	middle	Server	medium	medium	low	особливість	women	реліз 2
4	2	junior	Server	low	easy	low	особливість	women	реліз 2
5	2	junior	Reports	low	easy	low	особливість	women	реліз 2
6	1,5	middle	Settings	low	easy	low	дефект	women	реліз 2
7	3	junior	Settings	low	easy	High	дефект	women	реліз 2
8	1,5	middle	Settings	low	easy	High	особливість	women	реліз 2
9	1	senior	Reports	low	easy	High	особливість	women	реліз 3
10	0,5	senior	Reports	low	easy	low	особливість	women	реліз 3
11	1	senior	Reports	low	difficult	low	особливість	women	реліз 3
12	1,5	middle	Reports	blocking	medium	low	особливість	men	реліз 3
13	1	senior	Reports	low	easy	low	особливість	women	реліз 2
14	2	middle	Reports	blocking	medium	low	дефект	men	реліз 2
15	4	trainee	Reports	medium	medium	High	особливість	women	реліз 1
16	2,5	trainee	Settings	low	easy	High	дефект	women	реліз 1
17	1	senior	Archive	blocking	easy	medium	особливість	men	реліз 3
18	1,5	junior	Client	medium	easy	medium	покращення	women	реліз 2
19	0,5	senior	Client	medium	easy	medium	покращення	women	реліз 3
20	1,5	trainee	Client	medium	medium	High	покращення	women	реліз 1
21	2	senior	Client	high	medium	High	покращення	men	реліз 1
22	1,5	junior	Settings	low	medium	low	покращення	women	реліз 2
23	3	trainee	Settings	low	medium	low	дефект	women	реліз 2
24	2	junior	Settings	low	easy	medium	особливість	women	реліз 3
25	1,5	senior	Client	medium	medium	medium	особливість	women	реліз 1
26	1	senior	Client	medium	medium	medium	покращення	women	реліз 1
27	1,5	architect	Counter	low	medium	medium	дефект	women	реліз 1
28	1	architect	Counter	medium	medium	medium	дефект	women	реліз 1
29	0,5	architect	Counter	critical	easy	medium	покращення	men	реліз 1
30	1	architect	Archive	critical	easy	medium	особливість	men	реліз 3
31	1	architect	Settings	critical	easy	low	покращення	men	реліз 2
32	1,5	architect	Server	low	easy	High	покращення	women	реліз 1
33	0,5	architect	Counter	low	easy	low	особливість	women	реліз 2
34	1,0	architect	Server	low	medium	low	покращення	women	реліз 2

35	4	senior	Server	medium	easy	High	дефект	women	реліз 1
36	3	junior	Server	low	medium	High	дефект	women	реліз 1
37	1,5	middle	Client	critical	medium	High	дефект	men	реліз 3
38	2,5	middle	Client	medium	medium	High	особливість	women	реліз 1
39	2,5	senior	Archive	high	difficult	low	особливість	men	реліз 2
40	3,5	middle	Client	high	difficult	High	покращення	men	реліз 1
41	3	senior	Archive	high	difficult	low	особливість	men	реліз 2
42	4	middle	Archive	high	difficult	low	особливість	men	реліз 2
43	1,5	senior	Client	high	difficult	low	покращення	men	реліз 2
44	3	middle	Counter	blocking	difficult	High	особливість	men	реліз 1
45	2,5	middle	Counter	blocking	medium	High	особливість	men	реліз 1
46	1,5	senior	Settings	blocking	difficult	medium	дефект	men	реліз 3
47	3,5	middle	Settings	blocking	difficult	High	дефект	men	реліз 1
48	3	middle	Settings	blocking	difficult	High	дефект	men	реліз 1
49	2	senior	Settings	blocking	difficult	High	дефект	men	реліз 1
50	2	senior	Counter	high	difficult	low	дефект	men	реліз 2
51	1,5	senior	Counter	high	medium	low	дефект	men	реліз 2
52	1	architect	Counter	critical	medium	low	дефект	men	реліз 2
53	1	architect	Counter	high	difficult	medium	дефект	men	реліз 3
54	1	architect	Server	medium	medium	medium	особливість	women	реліз 3
55	0,5	architect	Server	medium	easy	medium	особливість	women	реліз 3
56	1,5	senior	Settings	blocking	easy	medium	дефект	men	реліз 1
57	1,5	middle	Settings	blocking	easy	low	особливість	men	реліз 2
58	3	trainee	Settings	medium	easy	High	особливість	women	реліз 1
59	1	middle	Client	medium	easy	High	особливість	women	реліз 1
60	1	senior	Client	medium	easy	medium	дефект	women	реліз 2
61	2,5	senior	Reports	blocking	easy	medium	особливість	men	реліз 2
62	1,5	middle	Reports	blocking	easy	medium	покращення	men	реліз 3
63	2	middle	Reports	blocking	medium	High	покращення	men	реліз 1
64	1	senior	Reports	blocking	medium	High	дефект	men	реліз 1
65	2,5	middle	Reports	high	easy	low	дефект	men	реліз 2
66	2,5	junior	Reports	high	difficult	low	покращення	men	реліз 2
67	3,5	middle	Reports	low	difficult	low	покращення	women	реліз 2
68	1,5	senior	Reports	high	difficult	medium	особливість	men	реліз 3
69	2	middle	Reports	critical	easy	low	дефект	men	реліз 2
70	2,5	trainee	Archive	critical	easy	low	дефект	men	реліз 2
71	3	trainee	Archive	critical	easy	High	особливість	men	реліз 1
72	0,5	middle	Archive	critical	easy	High	дефект	men	реліз 3
73	3,5	middle	Archive	critical	difficult	High	дефект	men	реліз 3

74	1	senior	Archive	critical	difficult	low	особливість	men	реліз 3
75	1	senior	Counter	critical	medium	medium	особливість	men	реліз 1
76	2	senior	Client	high	difficult	low	дефект	men	реліз 3
77	3,5	trainee	Client	critical	medium	low	покращення	men	реліз 3
78	4	trainee	Client	medium	medium	medium	дефект	women	реліз 3
79	2	middle	Client	high	medium	High	дефект	men	реліз 1
80	2	middle	Client	high	difficult	High	особливість	men	реліз 1
81	1	senior	Settings	medium	easy	medium	дефект	women	реліз 3
82	1,5	senior	Settings	high	difficult	High	дефект	men	реліз 1
83	1,5	senior	Counter	blocking	difficult	High	покращення	men	реліз 1
84	3	architect	Counter	blocking	difficult	low	особливість	men	реліз 2
85	3	architect	Client	blocking	difficult	low	особливість	men	реліз 2
86	1,5	senior	Client	blocking	difficult	medium	покращення	men	реліз 1
87	4	trainee	Server	blocking	difficult	medium	дефект	men	реліз 3
88	2	senior	Server	critical	difficult	low	покращення	women	реліз 1
89	4	junior	Server	high	difficult	low	дефект	men	реліз 2
90	2	middle	Server	bloking	medium	High	покращення	women	реліз 1
91	1,5	senior	Archive	medium	medium	High	дефект	women	реліз 1
92	2	senior	Archive	high	medium	High	особливість	men	реліз 1
93	2	senior	Archive	medium	medium	medium	особливість	women	реліз 3
94	3,5	junior	Archive	medium	medium	medium	дефект	women	реліз 2
95	2,5	middle	Server	critical	medium	medium	особливість	men	реліз 3
96	2,5	middle	Server	critical	medium	low	дефект	men	реліз 2
97	4	trainee	Server	critical	medium	medium	особливість	men	реліз 3
98	2,5	middle	Reports	critical	medium	High	особливість	men	реліз 1
99	2	middle	Reports	high	medium	High	особливість	men	реліз 2
100	2,5	middle	Reports	medium	difficult	High	особливість	women	реліз 2