Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя
(повне найменування вищого навчального закладу)
Факультет комп'ютерно-інформаційних систем і програмної інженерії
(назва факультету)
Кафедра комп'ютерних систем та мереж
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

магістра
(освітній ступінь))

на тему: **Методи і засоби тестування програмного забезпечення комп'ютерних систем з використанням алгоритмів машинного навчання**

Виконав: студент (ка) 6 курсу, групи СІм-61
спеціальності 123 «Комп'ютерна інженерія»
(шифр і назва спеціальності)

	(підпис)	Цісарук Д.А. (прізвище та ініціали)
Керівник	(підпис)	Луцків А.М. (прізвище та ініціали)
Нормоконтроль	(підпис)	Тиш Є.В. (прізвище та ініціали)
Завідувач кафедри	(підпис)	Осухівська Г.М. (прізвище та ініціали)
Рецензент	(підпис)	Дуда О.М. (прізвище та ініціали)

Тернопіль
2021

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя
(повне найменування вищого навчального закладу)
Факультет комп'ютерно-інформаційних систем і програмної інженерії
Кафедра комп'ютерних систем та мереж

ЗАТВЕРДЖУЮ
Завідувач кафедри *Осухівська Г.М.*
«_____» _____ 2021 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ

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

за спеціальністю *123 «Комп'ютерна інженерія»*
(шифр і назва спеціальності)

студенту *Цісаруку Дмитру Андрійовичу*
(прізвище, ім'я, по-батькові)

1. Тема проекту (роботи) *Методи і засоби тестування програмного забезпечення комп'ютерних систем з використанням алгоритмів машинного навчання*
Керівник проекту (роботи) *Луцків Андрій Мирославович, к.т.н., доц.*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
Затверджені наказом ректора від «*28*» *жовтня* 2021 року № *4/7-916*
2. Термін подання студентом завершеної роботи
3. Вихідні дані до роботи *Методи тестування програмного забезпечення, алгоритми і моделі машинного навчання, типи вимог до програмного забезпечення комп'ютерних систем*
4. Зміст роботи (перелік питань, які потрібно розробити)
Вступ. 1. Аналіз особливостей процесу тестування програмного забезпечення комп'ютерних систем 2. Розробка методів класифікації вимог та прогнозування дефектів ПЗ з використанням алгоритмів машинного навчання 3. Апробація методів машинного навчання при класифікації вимог та прогнозуванні дефектів програмного забезпечення комп'ютерних систем. 4. Охорона праці та безпека в надзвичайних ситуаціях. Висновки
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
1. Актуальність і мета дослідження. 2. Задачі дослідження, , об'єкт і предмет, наукова новизна і практична цінність дослідження. 3. Життєвий цикл тестування програмного забезпечення комп'ютерних систем. 4. Класифікація нефункціональних вимог у репозиторії PROMISE. 5. Алгоритм класифікації вимог до ПЗ та прогнозування дефектів програмного коду 6. Алгоритм побудови моделі та алгоритмів прогнозування дефектів ПЗ. 7. Ефективність алгоритмів машинного навчання при прогнозуванні дефектів ПЗ 8. Висновки

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Охорона праці та	*Осухівська Г.М.*		
безпека в надзвичайних	*Стадник І.Я.*		
ситуаціях			

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	*Аналіз особливостей процесу тестування програмного*	*28.10.2021-*	*виконано*
	забезпечення комп'ютерних систем	*13.11.2021*	
2.	*Розробка методів класифікації вимог та прогнозування*	*13.11.2021 –*	*виконано*
	дефектів ПЗ з використанням ML алгоритмів	*20.11.2021*	
3.	*Апробація методів машинного навчання при класифікації*	*21.11.2021 –*	*виконано*
	вимог та прогнозуванні дефектів програмного	*28.11.2021*	
	забезпечення комп'ютерних систем		
4.	*Охорона праці та безпека в надзвичайних ситуаціях*	*28.11.2021 – 02.12.2021*	*виконано*
5.	*Оформлення пояснювальної записки*	*03.12.2021- 06.12.2021*	*виконано*
6.	*Оформлення графічного матеріалу*	*07.12.2021- 11.12.2021*	*виконано*
7.	*Попередній захист кваліфікаційної роботи магістра*	*15.12.2021*	*виконано*
8.	*Захист кваліфікаційної роботи магістра*		

Студент _____ *Цісарук Д.А.*
(підпис) (прізвище та ініціали)

Керівник проекту (роботи) _____ *Луцків А.М.*
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Тема кваліфікаційної роботи: “Методи і засоби тестування програмного забезпечення комп’ютерних систем з використанням алгоритмів машинного навчання ”/ Кваліфікаційна робота// Цісарук Дмитро Андрійович // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп’ютерно-інформаційних систем та програмної інженерії, група СІм-61 // Тернопіль, 2021 // с. – 90, рис. – 11, табл. – 15, аркушів А1 – 8, додат. – 1, бібліогр. – 28.

Ключові слова: метод, засіб, тестування, програмне забезпечення, комп’ютерна система, алгоритм, машинне навчання.

Метою роботи є дослідження методів і засобів тестування програмного забезпечення та імплементація алгоритмів машинного навчання для автоматичної класифікації вимог та прогнозування дефектів програмного коду

У дослідженні проведено аналіз важливих понять, принципів і послідовності виконання процесів, що використовуються при проектуванні комп’ютерних систем, зокрема, термінологічні особливості у процесі імплементації програмного забезпечення на етапі тестування, що дало змогу зрозуміти і в подальшому визначити шляхи імплементації методів машинного навчання для підвищення ефективності виконання стадій життєвого циклу

Запропоновано метод класифікації вимог до програмного забезпечення, що базується на методах машинного навчання і використовує чотири алгоритми: метод опорних векторів, наївний байєсовий класифікатор, логістична регресія та метод найближчих сусідів, а також техніки опрацювання тексту: «мішок слів», TF-IDF і χ^2 . Розроблено метод прогнозування дефектів програмного забезпечення у комп’ютерних системах, що використовує 6 алгоритмів машинного навчання і три набори дефектів і дає змогу забезпечити та передбачити з високою імовірністю можливість появи помилок у програмних модулях з врахуванням метрик програмного коду.

ABSTRACT

The theme of the thesis: " Methods and tools for testing computer systems software based on machine learning algorithms " /Master thesis / Tsisaruk Dmytro Andriiovych / Ternopil Ivan Pul'uj National Technical University, Faculty of Computer Information Systems and software engineering, group CIm -61 // Ternopil, 2021// p. - 90, fig. – 11, table. – 15, Sheets A1 – 8, Add – 1, Ref. – 28.

Keywords: method, tool, testing, software, computer system, algorithm, machine learning.

The aim of the work is to study the methods and means of software testing and implementation of machine learning algorithms for automatic classification of requirements and prediction of software code defects

The study analyzes important concepts, principles and sequences of processes used in the design of computer systems, in particular, terminological features in the process of software implementation at the testing stage, which allowed to understand and further identify ways to implement machine learning methods to improve efficiency of life cycle stages

The method of classification of software requirements based on machine learning methods and using four algorithms is proposed: the method of reference vectors, naive Bayesian classifier, logistic regression and the method of nearest neighbors, as well as word processing techniques: "word bag", TF-IDF and χ^2.

The method for predicting software defects in computer systems has been developed, which uses 6 machine learning algorithms and three sets of defects and makes it possible to provide and predict with a high probability the possibility of errors in software modules taking into account software code metrics.

ЗМІСТ

ПЕРЕЛІК ОСНОВНИХ УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ І СКОРОЧЕНЬ

BoW	Bag of Words
TF-IDF	Term Frequency – Inverse Document Frequency
PCA	Principal Component Analysis
SVM	Support Vector Machine
KNN	K Nearest Neighborhood
БД	База Даних
КС	Комп’ютерна Система
ПЗ	Програмне забезпечення

ВСТУП

Актуальність теми. Технології проектування сучасних комп’ютерних систем (КС), автоматизованих комплексів, розумних пристроїв дають змогу ефективно реалізовувати складні і водночас зручні у використанні для кінцевого користувача продукти, що інтенсивно увійшли у повсякденне життя. Однак побудова таких комплексів потребує впровадження науково-обґрунтованих та перевірених на практиці математичних моделей, методів і засобів у процеси життєвого циклу як комп’ютерної системи в цілому, так і її компонентів зокрема.

Сучасні комп’ютерні системи передбачають проектування апаратної, програмної і комунікаційної складових. Тому доцільно окремо розглядати життєві цикли таких компонентів, що в комплексі формують процеси життєвого циклу комп’ютерної системи. «Мозком» будь-якої комп’ютерної системи є програмне забезпечення з відповідними імплементованими алгоритми опрацювання даних. Тому можна вважати, що забезпечення відповідності реалізованих властивостей до вимог програмного забезпечення (ПЗ) формують 80% успіху системи. Для забезпечення процесів валідації та верифікації КС і ПЗ зокрема, використовуються різні підходи, серед яких найбільш ефективними є методи і засоби тестування.

Процес тестування ПЗ орієнтований на перевірку функціональних та нефункціональних вимог, виявлення та усунення дефектів і як наслідок забезпечення якості програмного забезпечення. Варто відмітити, що сучасні технології тестування показують дуже хороші результати і дають змогу забезпечити відповідність КС очікуванням користувачів. Однак, для підвищення ефективності процесу тестування доцільним є впровадження методів та алгоритмів машинного навчання на етапі формування і класифікації вимог до ПЗ, а також прогнозування можливих дефектів з оцінкою параметрів масштабованості комп’ютерної системи, кількості стрічок коду, часу витраченого на процес тестування та ряду інших. Тому актуальною задачею при проектуванні комп’ютерних систем є обґрунтування, розробка та імплементація алгоритмів машинного навчання у процес тестування ПЗ.

Процес, моделі, методи і критерії тестування досліджуються багатьма вченими як в Україні (Г. Коваль, Т. Коротун, К. Лавріщева, В. Яцишин, О. Харченко та ін.), так і закордоном (T. Mayers, A. Patel, E. T. Barr, M. Harman, P. McMinn, M. Shahbaz). Основні результати їх досліджень забезпечують високу оптимізацію і точність результатів тестування, а також скорочують час даного процесу без втрати якості кінцевого продукту. Однак, враховуючи сучасні тенденції щодо розвитку та застосування алгоритмів машинного навчання, актуальним та доцільним є їхня імплементація у життєвий цикл процесу тестування ПЗ у КС.

Мета і задачі дослідження. Метою роботи є дослідження методів і засобів тестування програмного забезпечення та імплементація алгоритмів машинного навчання для автоматичної класифікації вимог та прогнозування дефектів програмного коду.

Для досягнення вказаної мети в роботі поставлено наступні **задачі**:

– аналіз методів, моделей, засобів та особливостей процесу тестування програмного забезпечення комп’ютерних систем;

– обґрунтування та формалізація факторів впливу, які необхідно врахувати при проведенні тестування ПЗ;

– аналіз та обґрунтування алгоритмів машинного навчання для імплементації у процес тестування ПЗ;

– розробка та обґрунтування методу класифікації вимог на етапі тестування програмного забезпечення;

– розробка та обґрунтування методу прогнозування дефектів програмного забезпечення;

– програмна реалізація запропонованих методів машинного навчання у процес тестування програмного забезпечення.

Об’єкт дослідження: процес тестування програмного забезпечення .

Предмет дослідження: алгоритми машинного навчання, моделі, методи і засоби тестування програмного забезпечення.

Методи дослідження: При розв'язанні задач дипломного проектування використано наступні методи: критичний аналіз – при дослідженні методів і засобів тестування програмного забезпечення комп'ютерних систем; машинне навчання – при побудові алгоритмів класифікації вимог та прогнозуванні дефектів; програмування – при реалізації алгоритмів машинного навчання; апробація – при верифікації результатів алгоритмів машинного навчання.

Наукова новизна отриманих результатів. Наукова новизна результатів дослідження полягає в наступному:

– уперше запропоновано метод класифікації вимог до програмного забезпечення, що базується на методах машинного навчання і використовує чотири алгоритми: метод опорних векторів, наївний байєсовий класифікатор, логістична регресія та метод найближчих сусідів, а також техніки опрацювання тексту: «мішок слів», TF-IDF і χ^2 , що в комплексі дало змогу забезпечити ефективність процесу класифікації вимог за групами функціональних і нефункціональних вимог, а нефункціональних за 11 підкласами вимог якості.

– уперше запропоновано метод прогнозування дефектів програмного забезпечення у комп'ютерних системах, що використовує 6 алгоритмів машинного навчання і три набори дефектів і дає змогу забезпечити та передбачити з високою імовірністю можливість появи помилок у програмних модулях з врахуванням метрик програмного коду.

Практичне значення одержаних результатів. Практичне значення одержаних результатів полягає у програмній реалізації алгоритмів машинного навчання при класифікації вимог на етапі тестування та при прогнозуванні дефектів програмного забезпечення.

Публікації. Результати кваліфікаційної роботи апробовані на X міжнародній науково - технічній конференції молодих учених і студентів «Актуальні задачі сучасних технологій» (24-25 листопада 2021 р.) Тернопільського національного технічного університету імені Івана Пулюя та на IX науково-технічній конференції Тернопільського національного технічного університету імені Івана Пулюя

«Інформаційні моделі, системи та технології» (8-9 грудня 2021 року) як тези конференцій.

1. Яцишин В.В., Шуптарський В.В., Цісарук Д.А. Алгоритми машинного навчання для сегментації користувачів у маркетингових комп'ютерних систем. Матеріали X міжнародної науково - технічної конференції молодих учених і студентів «Актуальні задачі сучасних технологій» (24-25 листопада 2021 р.) Тернопільського національного технічного університету імені Івана Пулюя. Тернопіль: ТНТУ. 2021. С. 145.

2. Луцків А.М., Цісарук Д.А., Шуптарський В.В. Аналіз життєвого циклу процесу тестування програмного забезпечення комп'ютерних систем. Матеріали IX науково-технічної конференції Тернопільського національного технічного університету імені Івана Пулюя «Інформаційні моделі, системи та технології» (8-9 грудня 2021 року). Тернопіль: ТНТУ. 2021. С. 142.

Структура роботи. Кваліфікаційна робота містить розрахунково-пояснювальну записку та графічний матеріал. До складу записки входить вступу, 4 розділи, загальні висновки, список використаних джерел і додатки. Обсяг роботи: розрахунково-пояснювальна записка – 90 арк. формату А4, графічна частина – 8 аркушів формату А1.

РОЗДІЛ 1
АНАЛІЗ ОСОБЛИВОСТЕЙ ПРОЦЕСУ ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ КОМП'ЮТЕРНИХ СИСТЕМ

1.1. Аналіз основних понять та особливостей процесу тестування ПЗ КС

Термінологічно «тестування програмного забезпечення» в контексті проектування комп'ютерних систем описує процес виявлення відповідності реальних одержаних результатів щодо очікувань від системи. Фактично, результати тестування показують загальну оцінку реалізації програмного забезпечення, тобто орієнтовані не тільки на виявлення «багів» і дефектів, але й на покращення, оптимізацію і підвищення якості продукту.

Інженери з якості повинні формувати задачі у відповідності до серйозності помилки. Добре протестоване і перевірене ПЗ забезпечує замовникам покращений продукт з врахуванням функціональних особливостей, вимог точності, безпеки даних та ін. Процес тестування ПЗ є ітераційним, що передбачає можливість його провадження навіть у випадку введення в експлуатацію.

Основна ціль, яку переслідує процес тестування, полягає у перевірці і підтвердженні комплектності і придатності до використання комп'ютерної системи. Комп'ютерна система та усі її компоненти повинні відповідати визначеним технічним вимогам.

Помилки і дефекти, виявлені під час тестування, повинні бути належним чином відображені у відповідних звітах. Окрім цього, тестувальники також мають переконатися в тому, що програмне забезпечення працює стабільно, без збоїв і лише тоді воно може бути реалізованим на ринку. Для досягнення цієї мети, особи, відповідальні за перевірку програмного забезпечення КС, повинні генерувати якісні тестові сценарії та формувати правильні звіти про виявлені проблеми.

Впровадження процесу тестування ПЗ у життєвий цикл КС дає змогу економити багато трудових ресурсів, грошей і часу для організацій, які займаються їхнім створенням і продажем. Можна сказати, що тестування забезпечує

оптимізацію бізнесу. Правильно організований та проведений процес тестування дозволяє підтвердити, що система чи продукт здатні правильно опрацьовувати дані і показувати адекватну поведінку за усіх передбачених і попередньо визначених умов, зокрема у різних типах операційних систем та у веб-браузерах. Це дає змогу покращити досвід користувачів і підвищити задоволеність клієнтів.

Всі вищезазначені заходи, які характерні для процесу тестування ПЗ, сприяють збільшенню доходів фірм-розробників комп'ютерних систем і дозволяють переконатися у тому, що ПЗ перед випуском на ринок є безпечним і функціонує належним чином. Базові поняття, які були введені у процес тестування ПЗ є важливими, оскільки помилки, збої та дефекти потенційно можуть призвести до збільшення витрат і небезпечних наслідків. Характерним для тестування є творчість та інтелектуальні знання членів команди, які забезпечують його перевірку.

Існують принципи, про які кожен тестувальник повинен знати під час перевірки програмного коду, візуального інтерфейсу, функціональності та ін. характеристик ПЗ. Тестування тісно пов'язане з різними проблемами, такими як масштабованість, зручність використання, безпека тощо, які відносяться до нефункціональних властивостей КС чи ПЗ.

У [1] проведено порівняльне дослідження поточного стану у сфері тестування ПЗ на ранніх етапах виконання проекту, описано та визначено цілі такого процесу, а також встановлено та проаналізовано різні рівні тестування програм.

Більше того, ця стаття зосереджується на тому, чому потрібно тестувати системи і які усталені термінологічні особливості, пов'язані з цим процесом [1].

У [2] розглядаються різні підходи до тестування ПЗ. Основна увагу при цьому зосереджена на методах тестування, зокрема, тестування методом «білої скриньки» і «чорної скриньки». Основний акцент статті зосереджено на тому, як дати можливість бізнесу визнати та усвідомити небезпеку використання програмного забезпечення [2].

У [3] обговорюються різні типи тестування, згадуються різні методи тестування ПЗ.

У [4] детально описано процес тестування, необхідність його проведення, цілі та основні принципи. Крім того, означено та обгрунтовано застосування різних підходів, такі як тестування коректності, продуктивності, надійності, безпеки, що дозволяють визначити відповідність нефункціональним вимогам, проаналізовано тестування методом «сірої скриньки».

У [5] розглянуто всі основні аспекти життєвого циклу тестування ПЗ (STLC) та обгрунтовано важливість звіту про помилки, виявлені під час тестування, детально пояснена структура та наповненість звітів про дефекти. Повідомлення про помилки, важливість та гарантія якості програмних додатків є основним акцентом статті [5].

У [6] досліджено різні стратегії тестування, а також проаналізовано ефективність інструментальних засобів при проведенні.

Виконання тест-кейсів у запланований і систематичний спосіб називають життєвим циклом процесу тестування. Життєвий цикл, в окремих термінах, можна викласти як зміну від однієї форми до іншої. Кожна сутність має свій життєвий цикл. Програмне забезпечення також є сутністю. Нижче наведено основні процеси життєвого циклу, які необхідні при розробці ПЗ.

1.2. Аналіз процесів життєвого циклу ПЗ КС

Базовим поняттям в індустрії розробки ПЗ є життєвий цикл, що передбачає виконання ряду взаємопов’язаних процесів. Найбільш важливими серед яких є проектування, імплементація і тестування, що в кінцевому результаті дозволяє випустити високоякісний програмний продукт.

Реалізація життєвого циклу повинна ефективно працювати в існуючій інфраструктурі фірми-розробника і бути орієнтованою на скорочення та ефективне використання ресурсів, як часових, так і трудових.

Це дає розробникам можливість вибору структури, яка є оптимальною під час розробки будь-якого продукту чи програми, а також формує базис для виконання проекту. Життєвий цикл пропонує можливості щодо формування повного уявлення про оцінку продукту або додатку, планування та послідовності виконання процесів, а також набір використовуваних стандартів. Іншими словами, можна сказати, що це один із механізмів відстеження стану проекту та його моніторингу.

Крім цього, життєвий цикл розробки ПЗ повинен надавати повну прозорість та видимість результатів зацікавленим сторонам, які залучені до команди розробників. Це призводить до збільшення швидкості розробки проекту, підвищення його якості та зменшення ризиків на кожному з етапів реалізації. Структуру процесів життєвого циклу розробки ПЗ при побудові КС показано на рис. 1.1.

Рис. 1.1. Основні процеси SDLC

Аналіз вимог є першою фазою життєвого циклу при виробництві будь-якого продукту, в тому числі і програмного. Основна мета даного процесу полягає в отриманні від замовника усієї необхідної для виконання проекту інформації. Обов'язковою умовою при цьому є взаємодія з клієнтом, що в подальшому забезпечить відповідність системи очікуванням споживача.

Бізнес-аналітик і менеджер проекту взаємодіють із замовником шляхом організації зустрічей та інтерв'ю з клієнтом і забезпечують збір необхідних для виконання проекту даних, наприклад, яка мета системи, хто буде кінцевим користувачем цього продукту тощо.

Перед створенням продукту необхідно досконало знати призначення продукту, що є надзвичайно важливим. Після збору всіх даних можна провести їх аналіз для того, щоб визначити, скільки часу знадобиться на розробку продукту, скільки ресурсів буде задіяно, які обмеження та які переваги майбутньої системи, а також деякі інші показники. На цьому етапі документ з вимогами має бути узгодженим, щоб розробники, а також замовники могли посилатися на цей нього в подальшому.

Формування техніко-економічного аналізу і, як результат, затвердження відповідного документу є необхідним для розробки комп'ютерної системи. Для підготовки цього документа виконуються різні перевірки, які утворюють відповідну специфікацію. Документ техніко-економічного аналізу повинен містити такі пункти:

– можливість проекту бути завершеним у межах бюджету, який передбачений заздалегідь;

– можливість існуючої комп'ютерної системи підтримувати програмне забезпечення,

– можливість реалізації різних завдань чи операцій, які очікуються замовником чи клієнтом,

– здатність завершити проект у відповідності до складеного графіку.

Повний документ техніко-економічного аналізу повинен містити найбільш важливі фактори, які впливають на розробку проекту та розгортонання готового продукту.

На етапі дизайну ПЗ, проектуються всі функції, архітектура продукту, усі необхідні специфікації запропонованої системи. Архітектура ПЗ може бути побудована двома способами. Такими способами є дизайн високого рівня та дизайн низького рівня.

У проектуванні високого рівня вказується короткий опис архітектури, окреслюється функціональність, яка знадобиться для проектування модуля, обговорюється специфікація інтерфейсу, готується інформація про архітектуру та її схема, ідентифікація бази даних, деталі технології разом з архітектурною схемою.

Низькорівневе проектування описує структуру вхідних і вихідних елементів, спосіб формування повідомлення про помилку, вигляд інтерфейсів продукту, тип і розмір бази даних, проблеми залежностей. Ця інформація допоможе розробникам зрозуміти загальну комплексну архітектуру системи.

Фаза імплементації або написання програмного коду є найдовшим етапом у процесі SDLC. На даному етапі розробники пишуть код з використанням обраної мови програмування. Усі завдання поділяються на підзавдання або на модулі і призначаються різним розробникам. Розробники повинні написати код відповідно до інструкцій, під час якого мають бути врахованими різні функціональні вимоги та обмеження. Розробники повинні дотримуватися проектної документації.

На етапі тестування команда забезпечення якості та команда тестування перевіряють функціональність продукту. Після того, як частина функціональності є реалізованою, розробники розгортають його з метою тестування. Група контролю якості і тестування проводить перевірку частини або усього продукту на предмет відповідності поточних результатів очікуваним, які затверджено у документі з вимогами.

Якщо команда контролю якості та тестування виявить помилку у продукті, вони фіксують і позначають її для виправлення відповідному розробнику. Після цього, розробник виправляє дефект і надсилає код на повторну перевірку.

Якщо помилку усунуто, QA та команда тестування закриває її, якщо ні – помилка має статус відкритої (не виправленої). Такий цикл буде продовжуватися до тих пір, поки реалізована функціональність і продукт не стане безпомилковим.

Етап інсталяції і розгортання комп’ютерної системи починається тоді, коли всі помилки позначено як закриті. Це означає, що продукт досягає стадії, коли в системі немає дефектів, і коли всі фактичні результати відповідають очікуваним. На основі результатів зворотного зв'язку із замовником чи клієнтом, менеджером проекту приймається рішення про розгортання ПЗ.

Після успішного розгортання комп’ютерної системи, ПЗ переходить на етап супроводу і технічної підтримки, а клієнт починає ним користуватися. Користувачі системи під час її експлуатації надають відгуки про систему, з яких команда отримає ідею, яку функцію необхідно додати.

Тестування продуктів продовжується після розгортання системи. Можуть бути деякі сценарії, які взагалі не перевіряються. Виправлення помилок проводиться на кожному етапі. Після виправлення помилок здійснюється оновлення продукту. На етапі обслуговування виробнича команда також може додати нові функції.

1.3. Аналіз особливостей життєвого циклу тестування ПЗ

Життєвий цикл тестування програмного забезпечення комп’ютерних систем передбачає виконання різних комплексних заходів та видів діяльності, які формують деяку послідовність. Основна ціль підпроцесів життєвого циклу тестування полягає у забезпеченні та контролі якості прототипу системи та кінцевого програмного продукту. Структуру життєвого циклу процесу тестування програмного забезпечення показано на рис. 1.2.

Фаза тестування при реалізації програмного забезпечення відіграє важливу роль у загальному життєвому циклі комп’ютерної системи. Команді тестувальників потрібно спланувати багато заходів і кожна діяльність має бути орієнтована на забезпечення якості.

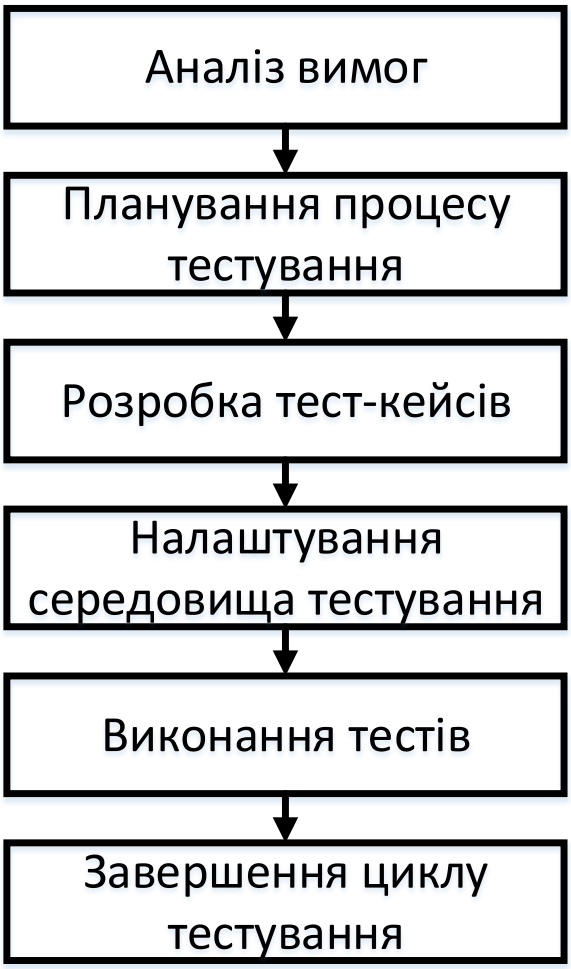

Рис. 1.2. Життєвий цикл тестування програмного забезпечення комп’ютерних систем

Характерною і необхідною ознакою кваліфікованої команди тестувальників є володіння інформацією щодо масштабу і повноти виконання подальших кроків щодо виявлення дефектів. Аналогічно, на ранній стадії тестування повинна бути можливість аналізу обсягу тестування продукту. Кожен документ щодо підготовки і проведення різних видів діяльності у процесі тестування повинен бути підготовлений належним чином і однаково трактуватись усіма учасниками проекту.

Якщо всі тестові сценарії охоплені належним чином у відповідності до вимог, то виконання тестового сценарію займатиме менше часу. Це допоможе виявляти помилки на ранній стадії.

При аналізі вимог (рис. 1.2), команда тестування та забезпечення якості вивчають та аналізують документ з вимогами. Вони намагаються досконало зрозуміти специфікацію і у випадку, якщо виникає нерозуміння або неоднозначне трактування пунктів документу, вони можуть взаємодіяти із зацікавленими сторонами. Це дозволяє більш детально зрозуміти та вирішити запити, пов’язані з документом специфікації вимог. Як тільки тестувальники правильно почнуть трактувати вимоги, то це дасть їм чітке уявлення про те, який тип тестування необхідно застосувати для перевірки системи. Тестери можуть налаштувати середовище для тестування.

На фазі планування визначають план і стратегії проведення тестування. Менеджер розраховує ресурси і час, необхідні для перевірки, розраховує оцінку вартості робіт. Команда тестування вирішує, яке середовище необхідне для провадження тестування та які їхні типи будуть проводитися, визначає ролі та межі відповідальності кожного члена команди.

Тестувальники в основному посилаються на документ з вимогами або плани тестування для створення тестових випадків (тест-кейсів). При створенні документа тестового сценарію слід враховувати всі умови та вимоги, перевірку яких необхідно забезпечити. У випадку наявності і доступності тестового середовища – на цій стадії створюються тестові дані.

Етап налаштування середовища тестування відіграє важливу роль у життєвому циклі тестування. Він визначає умови, за яких тестується ПЗ і є самостійним видом діяльності. Це означає, що тестувальники можуть почати виконувати цю діяльність паралельно із розробкою продукту. Результатом виконання цієї стадії є підготовлений перелік апаратного та програмного забезпечення тестового середовища. Тестове середовище організовує не тестер, а визначає команда розробників разом із замовником.

Фаза виконання тестів починається після завершення розробки тестових сценаріїв і налаштування тестового середовища. Тестувальники починають виконувати тестові випадки, і якщо у них виникає будь-який збій, вони фіксують і повідомляють про це у відповідному звіті. Команда розробників вносить корективи

в програмний код для виправлення помилки. Після цього відбувається повторна перевірка і якщо тест проходить успішно її позначають як виправлену, в іншому випадку – помилка або дефект матимуть мітку відкрита.

На етапі закриття циклу тестування проводиться оцінка різних видів діяльності, зокрема: термін виконання, якість, результат виконання, умови проведення тестів та якість ПЗ. Для цього формуються тестові матриці і звіт про завершення тестування. Звіт містить різні кількісні показники щодо виявлення та усунення помилок і дефектів у комп'ютерній системі. Ця інформація є одним з аргументів щодо передачі робочого продукту замовнику та відображає скільки дефектів виникла, і який їхній розподіл за критеріями критичності і пріоритетів.

1.4. Аналіз життєвого циклу помилок

Життєвий цикл помилок проходить через різні фази і починається тоді, коли тестувальник виявив нову проблему, а закінчується, коли ця проблема вирішена. Життєвий цикл помилок може відрізняється, оскільки залежить від типу проекту. На рис. 1.3 показано типовий життєвий цикл помилок ПЗ у комп'ютерних системах.

Рис. 1.3. Життєвий цикл помилки у ПЗ

Виходячи з представлення, показаного на рис. 1.3, стадії життєвого циклу помилки можна трактувати наступним чином:

– нова помилка – під час тестування продукту, тестувальники виявлять будь-яку помилку і повідомляють про неї.

– прийняття до виправлення – передбачає фіксацію помилки із передачею інформації про неї відповідному розробнику.

– аналіз помилки (мітка відрита помилка) – щоразу, коли тестер створює нову помилку, статус цієї помилки буде відкритим і коли розробник почне працювати над усуненням проблеми, мітка помилки може змінюватись відповідно до робочого процесу, який він виконує.

– виправлення – зміна мітки помилки у випадку успішного усунення дефекту розробником та повторним надсилання інформації про змін мітки до тестувальників.

– повторне тестування – розробник повертає помилку тестувальнику для повторної перевірки.

– перевірено – статус, що присвоюється помилці в процес повторної перевірки помилки щодо того, що всі кроки працюють належним чином, які згадуються в помилці.

– закриття – якщо тестовий випадок, який згадується в помилці, працює належним чином, тестер позначає статус помилки як закритий.

– повторне відкриття – якщо тестовий випадок, який згадується у помилці, не працює належним чином, тобто якщо тестувальник не отримує очікуваних результатів, він позначає статус помилки як повторно відкрита і знову призначає цю помилку для забезпечення відповідності.

– дублікат – наявність мітки говорить про помилку, яка вже існує чи існувала в процесі тестування і одна з них може бути позначена як дублікат.

– поточне усунення – випадок, коли розробник починає роботу над усуненням дефекту і позначає статус помилки як поточний.

– відкладеність – коли повідомлення про помилку не мають високого пріоритету і коли її можна виправити в наступному випуску.

– не є помилкою (дефектом) – якщо помилка, про яку повідомляється, неправильна або якщо помилка не є справжньою.

1.5. Класифікація методів тестування за способом його проведення

Після успішної імплементації програмного коду, тестувальникам необхідно перевірити функціональність системи. Методи тестування ПЗ за способом його проведення поділяються на дві категорії: статичне і динамічне.

1.5.1. Статичне тестування

Ручне тестування, або по-іншому статичне, є примітивним типом тестування, що передбачає виконання тестових випадків не автоматизованим шляхом. Будь-який продукт слід спочатку протестувати вручну, перш ніж виконувати автоматичне тестування. Ручне тестування допомагає знайти помилки в продукті, коли проект знаходиться на початковій стадії розробки. Даний вид перевірки використовується при тестуванні інтерфейсу користувача, особливо з візуальними аспектами, або коли проект є короткостроковим. Статичне тестування можна ефективно застосовувати у короткострокових проектах, коли написання сценарію займає багато часу.

Алгоритм виконання ручного тестування передбачає виконання наступних кроків:

Крок 1. Аналіз і розуміння специфікації вимог. Для того, щоб успішно провести ручне тестування, спочатку потрібно зрозуміти специфікацію вимог. Це робиться з метою визначення сутностей, які підлягають перевірці та яким чином відбувається класифікація потенційних проблем: як дефект чи помилка.

Крок 2. Написання тестових сценаріїв. Коли тестувальник зрозумів вимоги до системи, він може переходити до безпосереднього написання тестових сценаріїв, які допоможуть виконати послідовність дій, функцій та їхніх комбінацій для перевірки правильності функціонування окремих компонентів системи або продукту в цілому. Написання чітких та однозначно трактованих тестових випадків

дуже важливо, оскільки вони забезпечують ефективність процесу тестування та достатнього покриття тестами.

Крок 3. Виконання тестів. Як тільки завершено написання сценаріїв тестових випадків, наступним кроком є безпосереднє їх проведення у належним чином підготовленому тестовому середовищі. Під час цього кроку можна проводити моніторинг за результатами виконання тестових прикладів та позначати кожен тест однією з міток: успішний, невиконаний чи пропущений. Виконуючи ручне тестування, важливо вести записи та не ігнорувати факти щодо провалу тестів.

Крок 4. Фіксація інформації про помилку. У випадку виявлення помилки потрібно зробити відповідний запис у журналі для команди розробників про наявність дефекту. Написання хорошого звіту про помилку допомагає заощаджувати час на її усунення і повинен містити назву, кроки щодо подолання проблеми, очікувані та фактичні результати та будь-які додаткові дані, які допоможуть команді розробників зрозуміти помилку, наприклад, знімок екрана, відеозапис тощо.

Крок 5. Створення детального звіт про тестування. Після виконання усіх тестів корисно володіти статистичною інформацією, що описує результати тестуваннів, наприклад, скільки тестів було виконано успішно, скільки тестів було пропущено, скільки тестів було пройдено і скільки тестів було пропущено.

1.5.2. Динамічне тестування

При динамічному (автоматизованому) тестуванні використовуються відповідні CASE-засоби підтримки цього процесу. Тестові набори даних, у такому випадку, запускаються на засобах автоматизації, тоді як при ручному тестуванні тестувальники повинні сидіти і запускати тестові набори один за одним.

При динамічному тестуванні, тестувальникам хоч і необхідно написати тестові сценарії, однак воно швидше, ніж ручне, а випадкові перевірки не допускаються. Для проведення автоматизованих тестів залучаються члени команди, які є висококваліфікованими тестувальниками і мають хороші знання в

цій області, а також володіють відповідними знаннями мов програмування. Автоматизоване тестування часто відоме як тестування на валідацію.

При зміні цілей тестування або тестових сценаріїв повинні виконуватись модифікації програмного скрипта, що забезпечує перевірку валідності реалізованої вимоги. Тестування може бути проведеним на різних операційних платформах, які допомагають скоротити час його виконання. При автоматизованому тестуванні можуть бути використані такі його різновиди як: регресійне, навантажувальне і тестування продуктивності.

Оскільки, динамічне тестування проводиться відповідними автоматизованими інструментами, то він забезпечує більшу точність та адекватність результатів та є більш економічно ефективним і надійним у порівнянні з ручним.

З іншої сторони, динамічне тестування вимагає більших інвестицій в інструментальні засоби та в скіли інженерів з автоматизації. Окрім цього, налаштування середовища для тестування сценаріїв є менш складним.

Основні етапи автоматизованого тестування передбачають виконання наступних кроків:

Крок 1: Залучення експертів галузі щодо можливості впровадження засобів автоматизації процесу тестування.

Крок 2: Обґрунтування щодо правильного вибору інструментів для тестування.

Крок 3: Аналіз різних додатків і на основі його результатів вибір такого, який найкраще підходить для автоматизації.

Крок 4: Навчання інших учасників команди.

Крок 5: Створення основи для автоматизації.

Крок 6: Виконання розробленого плану.

Крок 7: Створення сценаріїв.

Крок 8: Фіксація помилок.

Крок 9: Обслуговування сценаріїв.

1.6. Висновки до розділу

У цьому розділі одержано основні наукові і практичні результати:

1. Проведено аналіз важливих понять, принципів і послідовності виконання процесів, що використовуються при проектуванні комп’ютерних систем, зокрема, термінологічні особливості у процесі імплементації програмного забезпечення на етапі тестування, що дало змогу зрозуміти і в подальшому визначити шляхи імплементації методів машинного навчання для підвищення ефективності виконання стадій життєвого циклу.

2. Проаналізовано життєвий цикл помилок у програмному забезпеченні, що дало можливість врахувати особливості їх виявлення та усунення на етапі тестування з врахуванням підходів та методів розробки комп’ютерних систем та імплементацією алгоритмів машинного навчання щодо прогнозування дефектів у програмних модулях.

3. На основі аналізу і класифікації методів тестування визначено необхідність впровадження методу машинного навчання щодо класифікації вимог до програмного забезпечення, що в перспективі дозволить використовувати їх при створенні сценаріїв тестування та формуванні звітів з тестування.

РОЗДІЛ 2
РОЗРОБКА МЕТОДІВ КЛАСИФІКАЦІЇ ВИМОГ ТА ПРОГНОЗУВАННЯ ДЕФЕКТІВ ПЗ З ВИКОРИСТАННЯМ АЛГОРИТМІВ МАШИННОГО НАВЧАННЯ

2.1. Підходи і стратегії тестування ПЗ

Стратегії тестування дуже важливі з точки зору успішної реалізації як самого програмного забезпечення, так і комп'ютерної системи в цілому. Вони здатні поєднувати методи проектування тестових сценаріїв і як наслідок забезпечувати виконання запланованих систематичних кроків для адекватної перевірки реалізованих вимог замовника.

Стратегії тестування ПЗ розробляються тестувальником та менеджером проекту, що в сукупності дозволяє сформувати чотири їх типи:

– «Модульне тестування (Unit Testing)»;
– «Інтеграційне тестування (Integrity Testing)»;
– «Системне тестування (System Testing)»;
– «Приймальне тестування (Acceptance TEsting)»;

Найменшим елементом програмного забезпечення, який можна перевірити, є модуль. Тому така стратегія називається модульним тестування. Даний вид перевірки відповідності реалізації вимог виконується командою розробників. У зв'язку з цим, кожен з девелоперів повинен володіти відповідними знання щодо проектування та організації програмного коду. Окрім цього, члени групи із забезпечення якості також можуть проводити цей тип тестування.

Модульне тестування є найпростішим, економічно ефективним, а також таким, яке не займає багато часу. Протягом періоду проведення цієї стратегії можуть ефективно застосовуватись декілька методів, як-от функціональне, структурне тестування тощо.

Модульне тестування не залежить від усієї системи, але при цьому проводиться індивідуальна перевірка деякого невеликої агрегатної конструкції.

Декілька інженерів можуть виконувати цей тип тестування одночасно і вирішувати наявні проблеми паралельно. Модульне тестування відноситься до тестування методом «білої коробки» і виконується на найнижчому рівні. При об'єднанні двох або більше модулів формується інтегрована структура або група.

Основна мета інтеграційного тестування полягає в об’єднанні невеликих атомарних модулів та проведенні їхньої перевірки в комплексі, тобто відбувається перевірка групи елементарних одиниць. Після кожного додавання нового блоку в існуючу групу тестувальник повинен виконувати повторний тест.

При безперервному виконанні цього типу перевірки існує велика імовірність щодо зменшення помилок при наступному застосуванні регресійного тестування, оскільки виявлені в процесі інтеграційного тестування дефекти вже будуть усунуті, а відповідно потрібно менше зусиль при проведенні регресійної перевірки.

Якщо до структури додається новий модуль, тестувальникам потрібно повторно перевіряти кожен тестовий випадок від початку до кінця. Інтеграційне тестування відноситься до стратегій тестування верхнього рівня.

Перевірка усієї архітектури ПЗ є основною метою системного тестування. У даній стратегії передбачено об’єднання всіх наявних модулів і проведення наскрізного тестування.

Фактично, системне тестування дозволяє встановити відповідність продукту затвердженим вимогам, а якість ПЗ може бути забезпечена шляхом виконання даного виду стратегії. Відповідно до зазначеного вище, усі функціональні вимоги повинні бути підтверджені системним тестуванням, що проводиться у середовищі розробників. Деякі інші типи тестування також виконуються під час системного тестування, зокрема: перевірка безпеки, здатності до відновлення, інтерфейсів користувача, сумісності тощо. Даний вид стратегії відповідає принципам тестування методом «чорної скриньки».

Приймальне (приймально-здавальне) тестування є дуже важливим видом тестування і його ще називають тестуванням завершального етапу. Така стратегія виконується перед тим, як остаточно доставити систему кінцевому користувачеві. Він перевіряє, чи відповідає продукт, розроблений організацією, всім заданим

критеріям і відповідним вимогам, які сформульовані замовником або клієнтом. Приймально-здавальне тестування відоме також як тестування забезпечення якості, завершальне, перевірочне чи валідаційне тестування, що виконується у середовищі користувача. Основна його мета полягає у тому, щоб перевірити, чи система функціонує належним чином відповідно до вказаних вимог, а також на цьому етапі не повинно виникати помилок.

Тестування ПЗ допомагає виявити помилки/дефекти, які наявні у ньому, а забезпечити якість стає набагато простіше, якщо виявляти їх на ранніх стадіях виконання проекту. Впровадження процесу тестування має на меті зробити програмне забезпечення стабільним і таким, що відповідає очікуваному результату. Вимоги клієнтів у даному процесі мають бути пріоритетними, а тестувальник повинен знати стратегії тестування, вміти писати ефективні тестові сценарії та фіксувати повідомлення про помилки.

Підсумовуючи вище викладений матеріал, можна сказати, що сучасні стандартні методи тестування і стратегії дають змогу доставити стабільний і надійний продукт кінцевому користувачеві. Для тестування програмного забезпечення тестувальникам потрібна технологія на кожному рівні життєвого циклу, щоб гарантувати, що кожна частина системи, а також ПЗ в цілому працює належним чином без поломок. Кожен рівень тестування стратегій тестування програмного забезпечення має своє значення.

Модульне тестування виконується для перевірки невеликих модулів або частин програмного забезпечення, інтеграційне – для тестування інтегрованих модулів програмного забезпечення, системне – виконується для оцінки всієї системи або ПЗ, а приймальне – виконується для оцінки відповідності системи вимогам.

Методи тестування програмного забезпечення допомагають розробляти тестові випадки та тестові сценарії для виконання ручного та автоматизованого тестування відповідно. Ці методи забезпечують процедуру перевірки окремої частини системи, а також тестування всієї системи.

2.2. Розробка методу класифікації вимог до ПЗ з імплементацією методів машинного навчання на етапі тестування

Вимоги до ПЗ зазвичай формуються у вигляді тексту. Класифікація тексту – це спроба організувати текстові документи за категоріями на основі властивостей та атрибутів, що належать кожному тексту. Це завдання широко розглядається як задача навчання з вчителем, яке визначається як ідентифікація категорій нових документів на основі ймовірності, запропонованої певним навчальним корпусом вже позначених (ідентифікованих) міткою документів [6].

Класифікація тексту використовується у багатьох областях, включаючи ідентифікацію спаму та категоризацію новин. Концепція може здатися простою, і, маючи невелику кількість документів, можна проаналізувати кожен документ вручну та отримати уявлення про категорію, до якої належить документ. На основі цих знань можна згрупувати подібні документи в категорії або класи. Однак при збільшенні кількості документів до кількох сотень тисяч або мільйонів, дана задача класифікації стає набагато складнішою діяльністю. Саме в цьому контексті використання методів векторизації та навчання із вчителем або без нього є корисними.

Класифікація документів є загальною проблемою, яка не обмежується лише текстом, але також може бути розширена на інші елементи, як-от музика, зображення, відео та інші медіа [2].

Завдання класифікації вимог до ПЗ, в тому числі і для використання на етапі тестування полягає у визначенні категорії, до якої належить дана вимога [3]. Вимоги поділяються на два категорійних типи: функціональні вимоги (FR), які описують поведінку або функції, які надає система, і нефункціональні вимоги (NFR), які включають атрибути (такі як зручність використання, безпека, конфіденційність тощо), або обмеження в програмі, що розробляється, або в процесі розробки програмного забезпечення [4].

Навіть якщо вимоги до програмного забезпечення добре відомі та добре описані, автоматична класифікація вимог, написаних природною мовою, на

функціональні вимоги та підкатегорії нефункціональних вимог залишається проблемою.

За даними [8], це особливо пов’язано з тим, що зацікавлені сторони, а також інженери з вимог використовують різні термінології та структури речень для опису одного і того ж типу вимог. Висока невідповідність у визначенні вимог робить автоматичну класифікацію більш схильною до помилок, тому проблема полягає в тому, щоб знайти оптимальні способи реалізації хорошої автоматизованої класифікації.

Крім того, така класифікація потрібна, оскільки ручна класифікація вимог до програмного забезпечення є трудомістким завданням, особливо на великих проектах з величезною кількістю вимог [6].

Тому у даній роботі пропонується метод покращення процесу класифікації вимог до ПЗ з використанням техніки векторизації тексту та методів, відомих як Bag of Words (BoW), TF- IDF [9], і χ^2. Крім цього, потрібно встановити, яка модель машинного навчання має найкращу продуктивність при вирішенні задач класифікації вимог.

Доцільність включення у дослідження методу χ^2 обґрунтовується тим, що багато результатів досліджень показують, що вибір ознак може покращити ефективність класифікації тексту [10], а також тому, що згідно результатів багатьох дослідженнях χ^2 є одним з найбільш ефективних алгоритмів [10].

Результати цього експерименту можуть допомогти розробникам, які потребують автоматизації процесу класифікацію вимог до ПЗ при проведенні тестування, вибрати методи та алгоритми, які найбільше допоможуть у цьому.

Для розв’язання задачі щодо класифікації вимог до ПЗ пропонується використати такі техніки векторизації, як BoW, TF-IDF і χ^2. В якості алгоритмів класифікації обрано SVM, KNN, MNB і логістична регресія (LR).

За допомогою цих порівнянь можна встановити і продемонструвати, який найкращий метод векторизації працює для представлення вимог до ПЗ і який є найкращим алгоритм класифікації вимог. При проведенні досліджень використовуються набори даних PROMISE_exp.

В загальному, вимога – це потреба, функціональність або характеристика системи. Існує принаймні три означення терміну «вимога» до ПЗ:

1. Умова або здатність, необхідна користувачеві для вирішення проблеми або досягнення мети.

2. Умова або здатність, якою повинна володіти система або компонент системи для виконання контракту, стандарту, специфікації або інших офіційно встановлених документів.

3. Документоване представлення умови або можливості.

Під час процесу розробки ПЗ чи КС необхідно виявляти вимоги зацікавлених сторін, документувати їх належним чином, підтверджувати та верифікувати, а також керувати ними протягом усього життєвого циклу системи [16].

Виявлення вимог є важливою діяльністю, що є частиною процесу розробки вимог, оскільки вона має виявити, що ПЗ має виконувати, завдяки чіткому усвідомленню побажань і потреб різних зацікавлених сторін і перетворити їх у вимоги [12, 13].

У [15] сказано, що основною діяльністю розробки вимог є виявлення вимог до системи, яка буде розроблена. Основою для виявлення вимог є знання, отримані під час системного аналізу, що містить джерела вимог, які потрібно проаналізувати та уточнити [16]. Щоб правильно документувати вимоги, їх поділяють на категорії, основні з яких: функціональні та нефункціональні вимоги.

Функціональна вимога – це вимога щодо результату поведінки, яка має забезпечуватися функцією системи. За даними IEEE та ін. [18], функціональні вимоги – це вимоги, які визначають функцію, яку система або компонент системи повинні бути здатні виконувати.

У [18] функціональна вимога описує функціональність, яка має бути доступною для користувачів системи, частково характеризуючи її поведінку як відповідь на запит. Цей тип вимоги не повинен згадувати будь-які технологічні проблеми, тобто в ідеалі функціональні вимоги повинні бути незалежними від аспектів проектування та реалізації [19].

Нефункціональні вимоги визначають бажані критерії якості системи, яку необхідно розробити, і часто впливають на архітектуру системи більше, ніж функціональні вимоги [16].

Інше визначення, згідно з [9], полягає в тому, що: нефункціональна вимога відповідає набору обмежень, накладених на систему, яку необхідно розробити, встановлюючи, наприклад, наскільки вона приваблива, корисна, швидка чи надійна.

У [10] нефункціональні вимоги означено як щось, що описує неповедінкові аспекти системи, фіксуючи властивості та обмеження, за яких система повинна працювати, і, згідно з [11], ця категорія вимог представляє собою набір необхідних загальних атрибутів системи, включаючи портативність, надійність, ефективність, людську інженерію, тестованість, зрозумілість та модифікованість.

Нефункціональні вимоги можна розділити на підкатегорії і їх іноді класифікують як вимоги до продукту, вимоги організації чи процесу та зовнішні вимоги [12, 13].

Різниця між нефункціональними та функціональними вимогами в програмній інженерії має залежати від того, «як» і «що» системи виконують або пропонують як ресурси [12]. Репозиторій PROMISE розглядає 11 підкатегорій для цього типу вимог, які показано у табл. 2.1.

Таблиця 2.1

Класифікація нефункціональних вимог у репозиторії PROMISE

Тип нефункціональної вимоги	Опис
Доступність (Availability)	Описує, наскільки ймовірно, що система доступна для користувача в певний момент часу.
Відмовостійкість (Fault tolerance)	Ступінь, до якої система, продукт або компонент працюють належним чином, незважаючи на наявність апаратних чи програмних збоїв.

Продовження табл. 2.1

Тип нефункціональної вимоги	Опис
Сертифікати та ліцензії (Legal & Licensing)	Сертифікати або ліцензії, які повинна мати система.
Естетичне оформлення (Look&Feel)	Описує стиль зовнішнього вигляду продукту.
Ремонтопридатність (Maintainability)	Ступінь ефективності, з якою продукт або система може бути модифікована
Оперативність (Operability)	Ступінь володіння атрибутами, які полегшують керування продуктом або системою
Продуктивність (Productivity)	Продуктивність відносно кількості ресурсів, що використовуються за визначених умов.
Портативність (Portability)	Ступінь ефективності, з якою система, продукт або компонент можуть бути перенесені з одного середовища в інше.
Масштабованість (Scalability)	Ступінь, до якого продукт або систему можна ефективно адаптувати для різних апаратних засобів, програмного забезпечення або інших операційних чи виробничих середовищ.
Безпека (Security)	Ступінь, до якої продукт або система захищає інформацію та дані, щоб користувача чи інші системи мали ступінь доступу до даних, що відповідає їх типам і рівням авторизації.
Зручність використання (Usability)	Ступінь, до якої продукт або систему можуть використовувати певні користувачі для досягнення визначених цілей з ефективністю та задоволеністю у визначеному контексті використання.

Машинне навчання передбачає створення математичних моделей, які допомагають зрозуміти дані [14]. У даний час воно використовується в кількох контекстах, таких як: аналіз зображень продуктів на виробничій лінії для їх автоматичної класифікації [15], виявлення пухлин під час сканування мозку [16], автоматична класифікація новинних статей [17], виявлення кредитних шахраїв [18], створення чат-ботів або персональних помічників [19] та інші.

Останнім часом використання машинного навчання (ML) у програмній інженерії вивчається як для управління, так і для розробки програмного забезпечення [7].

Для проведення цих досліджень сховища даних, що стосуються процесу розробки ПЗ, такі як обговорення на форумах, історія обслуговування, відгуки та коментарі користувачів у соціальних мережах, стали потужним джерелом даних для використання у машинному навчанні, у поєднанні з аналізом тексту для отримання корисних даних.

У [12] зібрали велику кількість досліджень, пов’язаних із зв’язком між програмною інженерією та машинним навчанням. Автори виявили, що ML впливає на діяльність із виявлення, аналізу, перевірки та управління вимогами. У даному дослідженні буде використовуватися нормалізація тексту, методи виділення ознак та алгоритми машинного навчання для створення моделей класифікації вимог до ПЗ з використанням бази даних PROMISE_exp [3].

2.2.1. Нормалізація тексту вимог

Нормалізація тексту визначається як процес, який складається з серії кроків, які слід виконати, щоб усунути конфліктність, очистити і стандартизувати текстові дані і перетворити їх у форму, яку можуть використовувати інші системи опрацювання природної мови (NLP) та аналітичні системи та програми як вхідні дані [12].

Одним із таких кроків є токенізація, яка складається з поділу тексту на список лексем, причому ці лексеми можуть бути реченнями або окремими словами,

залежно від вибору дослідника. Часто сам токен також є частиною нормалізації тексту.

Окрім токенізації, процес нормалізації забезпечується й іншими методами, такими як: перетворення регістру, виправлення орфографії, видалення невідповідних слів та інших непотрібних термінів, таких як артиклі та займенники, створення коренів і лематизація.

У даному дослідженні база даних, що містить вимоги до програмного забезпечення, пройшла процес нормалізації, коли документи в корпусі були токенізовані, усі тексти були перетворені в нижній регістр, невідповідні слова були видалені, а сполучені слова в часах дієслів були перетворені в їх оригінальні форми за допомогою лематизації.

2.2.2. Векторизація тексту

Алгоритми машинного навчання працюють у числовому просторі ознак, що передбачає представлення даних у вигляді двовимірного масиву, де рядки є екземплярами, а стовпці — ознаками [7]. Щоб реалізувати машинне навчання над текстом, потрібно перетворити екземпляри (вимоги), документи, у векторні представлення, щоб мати можливість оперувати у просторі числового представлення.

Процес кодування документів у числовому просторі ознак називається виділенням ознак або, простіше кажучи, векторизацією і є важливим першим кроком до аналізу з урахуванням мови [7].

Векторна модель простору – це концепція та модель, яка дуже корисна, якщо працюють з текстовими даними. За даними [2], VSM дуже популярний у пошуку інформації та ранжуванні документів, що представляє собою алгебраїчну та математичну модель опису текстів у вигляді числових векторів. До них належать: Bag of Words [20], TF-IDF [21] та Averged Word Vectors [22].

Bag of Words, мабуть, є однією з найпростіших, але найпотужніших методів добування ознак із текстових документів [2]. Суть цієї моделі полягає в перетворенні текстових документів у вектори таким чином, щоб кожен документ

перетворювався на вектор, який представляє частоту всіх окремих слів, присутніх у векторному просторі документа для конкретного документа [2].

Використовуючи BoW, вимога «j» виражається вектором $X_j = (x_{1j}, x_{2j}, \dots, x_{ij} \dots, x_{nj})$, у якому x_{ij} – позначає вагу ознаки i у вимозі j і n – кількість термінів у словнику.

Після цього класифіковані вручну вимоги, які виражаються векторами, формуються як вхідні дані алгоритмів машинного навчання з вчителем, а отже, використовуються для навчання класифікаторів.

TF-IDF – це модель, яка поєднує у собі дві метрики: значення частоти входження терміну у конкретному документі та оберненої частоти документа для кожного терміна, яка обчислюється шляхом ділення загальної кількості документів корпусу на частоту документа для кожного терміну. Після цього застосовується логарифмічне масштабування результату.

При аналізі вимог до ПЗ, обернену частоту документа можна математично представити такою формулою:

$$idf_i = \log \frac{\text{загальна кількість вимог}}{\text{загальна кількість вимог з терміном } i} \quad (2.1)$$

Поєднуючи ці дві метрики, вектор ознак TF-IDF можна математично визначити як:

$$TF\_IDF\ (term_{ij}) = tf_{i,j} * idf_i, \quad (2.2)$$

де tf – частота вживання терміну, а idf – частота, обернена до частоти документа для терміну i та документа j.

Процес векторизації тексту у деяких випадках генерує слабкі інформативні ознаки для процесу класифікації, які можуть не допомогти або зашкодити продуктивності алгоритмам машинного навчання.

2.2.3. Обґрунтування вибору ознак вимог та алгоритмів класифікації вимог до ПЗ

Обґрунтування вибору ознак вимог можна використовувати як наступний крок у процесі представлення текстових документів з метою зменшення оригінального простору без шкоди для точності класифікації.

Процес підготовки даних, де видаляють атрибути, які не надають корисної інформації для вирішення поставленої задачі дозволяють зменшити використання апаратних ресурсів, забезпечити ефективність обчислень, а також зберегти мережеві ресурси на етапі навчання та для кожного майбутнього використання класифікатора [13].

При виборі ознак вимог до ПЗ пропонується дотримуватися принципу, що слова, які існують у більшій кількості класів, містять менше інформації про клас, а отже, менш важливі для класифікації.

χ^2 є загальним статистичним тестом, який вимірює відхилення від очікуваного розподілу, якщо припустити, що наявність ознаки насправді не залежить від значення класу [13]. Дана метрика вимірює відсутність незалежності між термом t і класом t. Для обчислення значення χ^2 використовується формула:

$$\chi^2(t,c) = \frac{N*(AD-CB)^2}{(A+C)*(B+D)*(A+B)*(C+D)} \quad (2.3)$$

де N – загальна кількість документів;

A – кількість випадків, коли t і c зустрічаються разом;

B – кількість разів, коли t існує без c;

C – кількість разів, коли c не містить t;

D – кількість випадків, коли ні c, ні t не зустрічаються.

Машинне навчання включає чотири основні [11]:

– «навчання з вчителем (supervised learning)» ;
– «навчання без вчителя (unsupervised learning)»;
– «часткове навчання (semi-supervised learning)»;

– «навчання з підкріпленням (reinforcement learning)».

При частковому навчанні, навчальний набір, який використовується для подачі на алгоритм, містить бажані результати, які називаються мітками.

З іншого боку, при навчанні без вчителя, навчальні дані не позначаються, і алгоритм повинен навчатися без попередньої допомоги.

Напівконтрольовані алгоритми використовують частково позначені бази, і під час навчання за допомогою підкріплення система може спостерігати за навколишнім середовищем, вибирати й виконувати дії, а також отримувати винагороду натомість.

Система навчання, яку в цьому контексті називають агентом, може спостерігати за навколишнім середовищем, вибирати й виконувати дії, а також отримувати винагороду (або штрафи) [15].

При класифікації вимог до ПЗ запропоновано використати алгоритми навчання з вчителем, до яких належать:

1. Метод найближчих сусідів – ґрунтується на принципі, що екземпляри в наборі даних, як правило, існуватимуть у безпосередній близькості від інших екземплярів, які мають подібні властивості. Алгоритм класифікує нові дані, обчислюючи відстань до вже існуючих екземплярів всередині бази даних, потім вибирає найближчі k екземплярів і обчислює їх середнє значення для задач регресії або отримує мітку класу.

2. Метод опорних векторів – машини опорних векторів є особливо потужним і гнучким класом контрольованих алгоритмів, що використовуються для розв’язання задач класифікації та регресії. Даний метод є потужним та універсальним засобом представлення моделі машинного навчання, що здатна виконувати лінійну або нелінійну класифікацію, регресію і навіть виявлення викидів. Алгоритм виконує класифікацію шляхом побудови лінійної гіперплощини з максимальним запасом, яка розділяє два класи. Завдяки цьому запасу є мало можливостей відокремити дані від вибірки, і, таким чином, є мало шансів помилково класифікувати нові екземпляри.

3. Логістична регресія – зазвичай використовується для оцінки ймовірності того, що екземпляр належить до певного класу (наприклад, яка ймовірність того, що цей лист є спамом?). Логістичну регресію можна визначити як клас регресії, в якому незалежна змінна використовується для прогнозування залежної змінної. Вона називається бінарною логістичною регресією, коли залежна змінна має дві можливі мітки класу. У випадку, коли залежна змінна може належати більше, ніж до одного з двох класів, то це називають мультиноміальною логістичною регресією. У випадку класифікації вимог до ПЗ при визначенні функціональних та нефункціональних вимог використовується бінарна логістична регресія., а мультиноміальна логістична регресія – для класифікації нефункціональних вимог за підкатегоріями.

4. Мультономіальний наївний Байєсовий класифікатор – генеративна модель оцінює умовну ймовірність приналежності даних до класу. Зокрема, при використанні цього класифікатора припускають, що вхідні ознаки є незалежними одна від одної (умовна незалежність). Мультиноміальний наївний байєсівський класифікатор є спеціалізованою версією базового класифікатора, який в основному використовується для класифікації документів і текстів [15].

2.2.4. Алгоритм класифікації вимог до ПЗ

Алгоритм класифікації вимог для подальшого їх використання на етапі тестування ПЗ включає чотири основні кроки, які показано на рис. 2.1:

1. Нормалізація – перший крок очищення даних. Усі невідповідні слова, наприклад займенники та артиклі вилучаються. Змінні дієслова та іменники перетворюються на їх кореневу форму.

2. Векторизація або добування ознак – на цьому етапі формується корпус вимог до програмного забезпечення, шляхом перетворення тексту у числові вектори та представляють інформацію, що міститься в цих вимогах. У даному дослідженні використовується BoW і TF-IDF для виконання цього перетворення.

3. Класифікація – на цьому кроці вектори, отримані на кроці 2, використовуються для навчання та прогнозування моделі класифікації чотирьох

алгоритмів: метод опорних векторів, мультиноміальний баєсовий класифікатор, метод найближчих сусідів і логістична регресія.

4. Оцінка ефективності класифікації – це останній етап класифікації, де результати прогнозованих міток вимог та їх реальні мітки використовуються для обчислення продуктивності процесу класифікації.

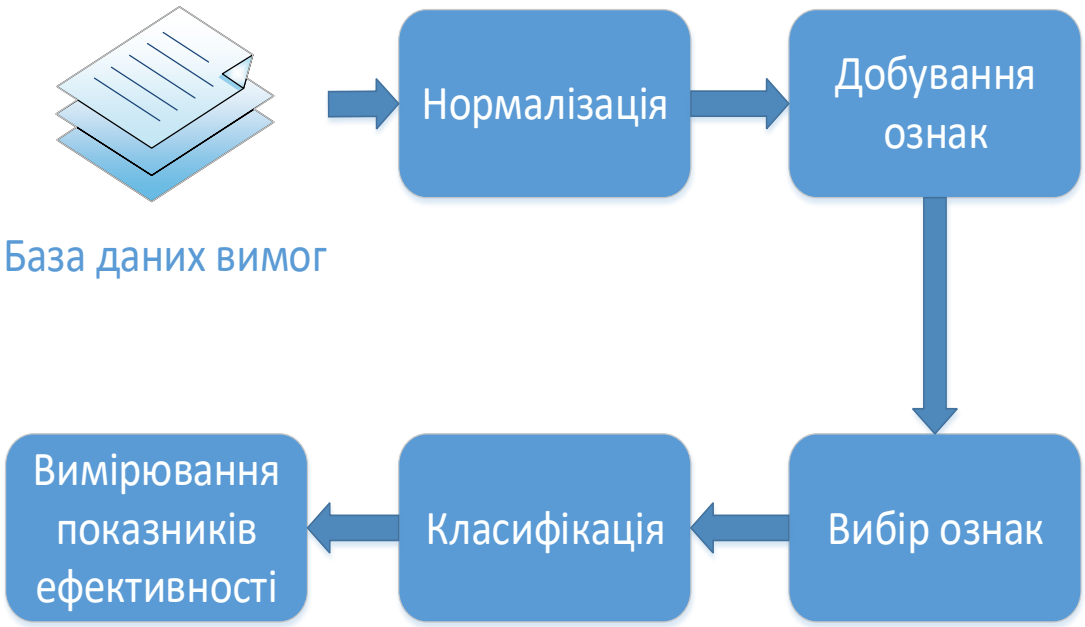

Рис. 2.1. Алгоритм класифікації вимог до ПЗ представлених у текстовому вигляді

Таким чином, обгрунтовано та формально представлено метод класифікації вимог до ПЗ з використанням алгоритмів машинного навчання при проведенні тестування ПЗ у комп'ютерних системах. Наступний крок полягає в імплементації алгоритмів машинного навчання для прогнозування дефектів.

2.3. Розробка методу прогнозування дефектів у програмному забезпеченні

Прогнозування помилок і дефектів у програмному забезпеченні, а також визначення імовірності їх появи, протягом довгого часу розглядалися як критична проблема для технологічної галузі та професіоналів у галузі розробки ПЗ.

При застосуванні традиційних методів виявлення і прогнозування дефектів обов’язковим є попередній досвід тестувальника і здатність аналізувати програмний код всередині комп’ютерної системи.

Автоматизовані моделі прогнозування і відновлення програмного забезпечення дозволяють значно підвищити якість прогнозування та відновлення збоїв програмного забезпечення за допомогою методів машинного навчання. Така здатність дозволяє програмному забезпеченню працювати більш ефективно і зменшувати кількість несправностей, час і вартість розробки.

У роботі пропонуються моделі прогнозування дефектів програмного забезпечення з використанням методів машинного навчання, які можуть дозволити передбачити імовірну появу помилок при тестуванні. Крім того, рекомендовано застосовувати різні метрики для оцінки ефективності моделі, такі як десятикратні методи перехресної перевірки, точність, запам’ятовування, специфічність, вимірювання f1, і точність.

Широка сфера технологій розробки ПЗ і різноманітність додатків формує складність проведення процесів їх моніторингу, підтримки та управління як для розробників, так і для клієнтів-замовників. Крім того, четверта промислова революція передбачає використання методів і моделей штучного інтелекту, а індустрія програмного забезпечення є однією з перспективних галузей сучасності, що характеризується постійною трансформацією шляхом автоматизації великої кількості технологій.

Розмір і складність сучасного ПЗ зростає день у день. У результаті інженери програмного забезпечення постійно борються з помилками починаючи з ранньої стадії виконання проекту. Класифікація помилок і дефектів ПЗ важлива у режимі реального часу, а зусилля та вартість пошуку дефектів, що приховуються в програмі, швидко зростають. Це формує актуальність досліджень щодо розробки автоматизованих моделей прогнозування помилок для виявлення дефектів. Якщо дефекти програмного забезпечення виявлені до випуску програмного забезпечення, то це може допомогти розробнику легко виправити помилки у дефектних модулях.

Прогнозування несправностей ПЗ за допомогою методів машинного навчання є найпомітнішим випадком використання серед дослідників і співтовариства програмного забезпечення [12].

Найсучасніші алгоритми машинного навчання були застосовані для пошуку модулів з дефектами у програмних додатках, а також для дослідження та створення ефективних рішень для споживачів [13].

У даній роботі пропонується використати шість найпопулярніших класифікаторів, які входять до алгоритмів машинного навчання. Усі вибрані методи класифікації застосовуються до різних реальних прикладних наборів даних пов'язаних з прогнозуванням помилок і дефектів у програмах.

Метою цього дослідження є обґрунтування необхідності щодо використанням шести класифікаторів і побудова рекомендацій щодо автоматизованого підходу до вирішення проблем всередині ПЗ з використанням трьох наборів даних – JM1, CM1, PC1 від PROMISE.

Ці набори даних були використані у науковій роботі [19], а також у [20] зібрано модель прогнозування програмних збоїв та виявлення дефектів ПЗ за допомогою алгоритмів машинного навчання. У цих публікаціях використано інформацію про 8 проектів і взято 19 метрик CK та McCabe для побудови моделі прогнозування. У даному дослідженні пропонується використати 22 атрибути для побудови моделі автоматизованого прогнозування несправностей. У табл. 2.2 наведено обрані атрибути з наборів даних про дефекти ПЗ, включаючи 21 незалежну метрику і одну, що містить інформацію про результати, тобто наявність або відсутність дефекту.

Таблиця 2.2

Метрики дефектів ПЗ при прогнозуванні

№ з/п	Метрика	Тип
1	Кількість стрічок коду	МакКабі
2	Цикломатична складність	МакКабі

Продовження табл. 2.2

№ з/п	Метрика	Тип
3	Основна складність	МакКабі
4	Складність дизайну	МакКабі
5	Оператори та операнди за Холстедом	Холстед
6	Об’єм ПЗ за Холстедом	Холстед
7	Довжина програмного коду за Холстедом	Холстед
8	Складність за Холстедом	Холстед
9	Інтелектуальність за Холстедом	Холстед
10	Розумові витрати програміста на створення коду за Холстедом	Холстед
11	Оцінка часу за Холстедом	Холстед
12	Кількість стрічок коду за Холстедом	Холстед
13	Кількість коментарів за Холстедом	Холстед
14	Кількість порожніх стрічок за Холстедом	Холстед
15	Кількість коментарів та операторів вводу/виводу	Інший
16	Кількість унікальних операторів	Пряме вимірювання
17	Кількість унікальних операндів	Пряме вимірювання
18	Загальна кількість операторів	Пряме вимірювання
19	Загальна кількість операндів	Пряме вимірювання
20	Кількість гілок коду	Пряме вимірювання
21	Кількість дефектів за Холстедом	Холстед
22	Дефекти	Наявні або відсутні

У цьому дослідженні пропонується використання наборів даних JM1, CM1, PC1, які були реалізовано мовою C. У табл. 2.3 наведено деталі всіх наборів даних з їх характеристиками.

Таблиця 2.3

Характеристики наборів даних

№ з/п	Набір даних	Пропущені значення	Кількість елементів	Розподіл за класами	
				Мітка «+»	Мітка «-»
1	JM1	Відсутні	10885	8779 (80,65%)	2106 (19,35%)
2	CM1	Відсутні	498	49 (9,83%)	449 (90,16%)
3	PC1	Відсутні	1109	1032 (93,05%)	77 (6,94%

При розробці методу прогнозування дефектів ПЗ пропонується використати наступні методи машинного навчання, які в комплексі формують модель прогнозування дефектів у ПЗ:

– дерева прийняття рішень;
– метод найближчих сусідів;
– логістична регресія;
– наївний байєсівський класифікатор;
– випадкові ліси;
– метод опорних векторів.

Запропонована модель прогнозування дефектів програмного забезпечення може бути впроваджена на початковому етапі життєвого циклу. Вона передбачає, що інтегрований процес розробки ПЗ з моделлю прогнозування імплементуються після того, як вхідні дані з аналізу вимог потрапляють у модель прогнозування.

При розробці ПЗ з використанням моделі прогнозування дефектів, діаграми високого рівня та рівня деталізації архітектури є обов’язковими для побудови масштабованої моделі прогнозування. Основною фазою розробки моделі виявлення і прогнозування дефектів є фаза фізичного проектування і тестування, аналіз модулів дефектів для забезпечення автоматизованого відновлення

програмних систем на основі знань. На рис. 2.2 показано імплементацію та місце моделі прогнозування дефектів ПЗ комп'ютерних систем у життєвому циклі системи.

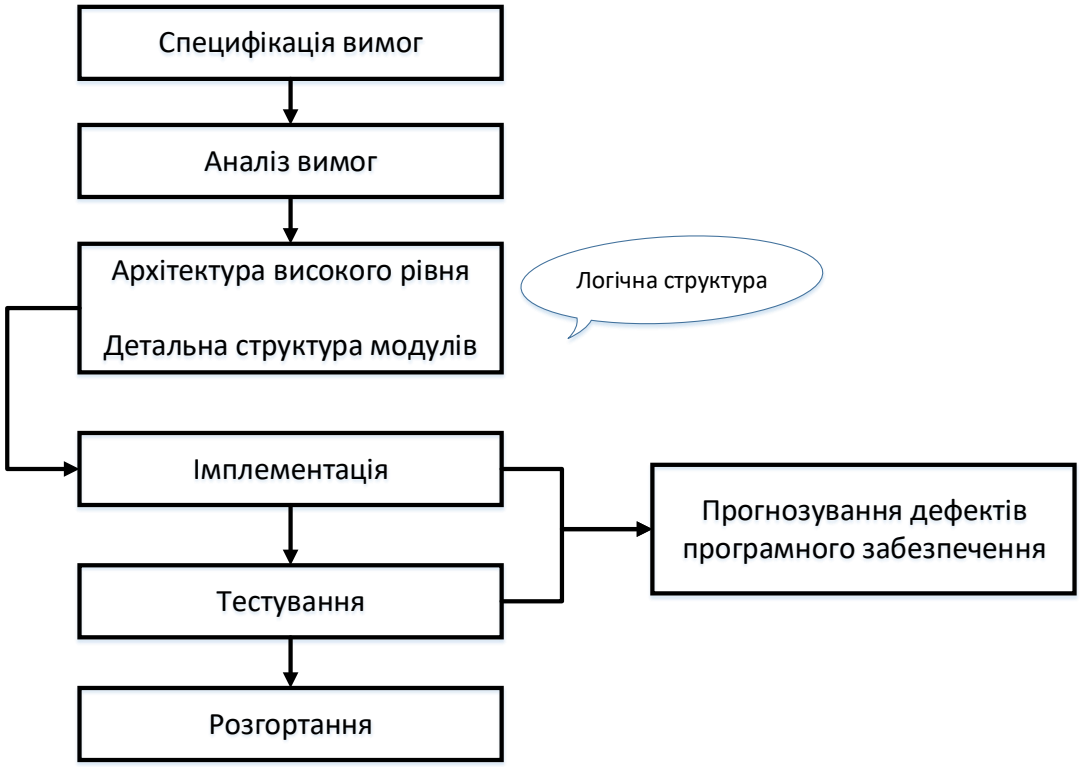

Рис. 2.2. Імплементація моделі прогнозування дефектів у життєвому циклі ПЗ

Далі необхідно визначити критерії ефективності моделей класифікації вимог та прогнозування дефектів ПЗ, а також визначити процедуру сумісного їх використання при реалізації комп'ютерних систем.

2.4. Метрики ефективності алгоритмів машинного навчання та імплементація процедури запропонованих методів при тестуванні ПЗ

Оскільки, як у випадку класифікації вимог до ПЗ, так і при побудові моделі прогнозування дефектів необхідно оцінити їхню ефективність, то доцільним є застосування відповідних метрик, щоб кількісно виразити показники якості цих моделей.

Після створення моделі прогнозування дефектів і класифікації вимог можна застосовувати провести тести, щоб передбачити результати у наборах даних про помилки ПЗ.

В якості метрик бінарної класифікація можна обрати наступні:

– матриця помилок – дозволяє визначити кількість правильно визначених результатів («True Positive = TP»), кількість неправильних результатів («True Negative = TN»), кількість правильних результатів з неправильною міткою («False Positive = FP»), кількість неправильних результатів з неправильними мітками («False Negative = FN»);

– відносна точність («Precision»);

– абсолютна точність («Accuracy»);

– F1 – метрика;

– конкретність («Specifity»).

У табл. 2.4 наведено формули, за якими виконується обчислення представлених вище метрик.

Таблиця 2.4

Метрики бінарної класифікації

Метрики	Математична формула
Точність («Accuracy»)	$\frac{TP + TN}{TP + FP + TN + FN}$
Точність («Precision»)	$\frac{TP}{TP + FP}$
Повнота («Recall»)	$\frac{TP}{TP + FN}$
F1-метрика	$\frac{2 * (Recall * Precison)}{(Recall + Precision)}$
Конкретність («Specificity»)	$\frac{TN}{TN + FP}$

На рис. 2.4 показано процедуру імплементації запропонованих методів бінарної та мультиноміальної класифікації вимог до ПЗ, а також прогнозування дефектів ПЗ при проектуванні комп'ютерних систем.

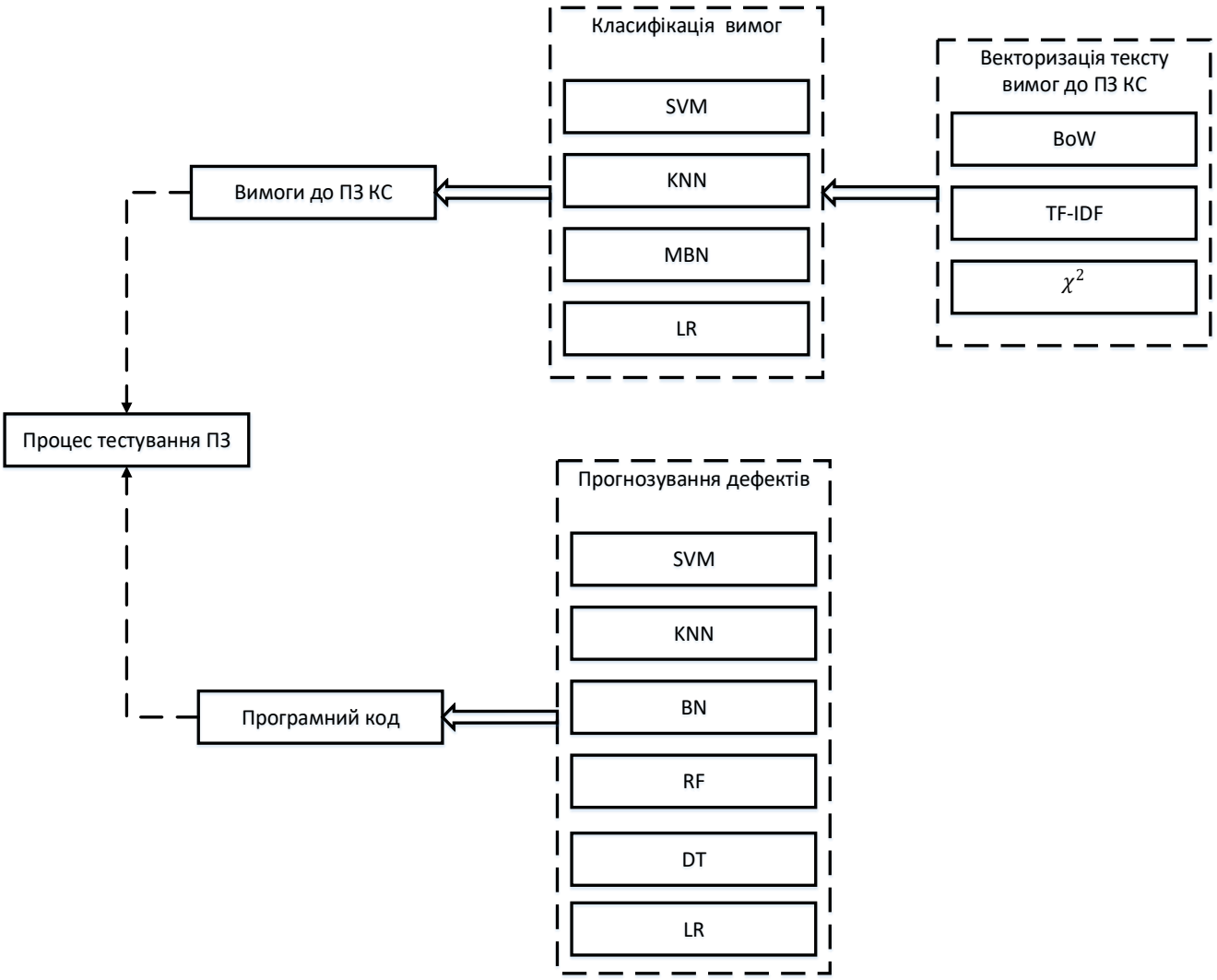

Рис. 2.4. Процедура імплементації методів машинного навчання у процес тестування ПЗ

Як видно з рис. 2.4, імплементація методів машинного навчання на етапі тестування відбувається шляхом визначення і класифікації вимог на функціональні та нефункціональні, а ті в свою чергу на підхарактеристики. Це дає змогу наперед визначити стратегії тестування, а також методи тестування для перевірки відповідних класів вимог. Процедура автоматичної класифікації вимог виконується

на ранній стадії життєвого циклу, що дає змогу підвищити ефективність їхньої перевірки.

Прогнозування імовірних дефектів і помилок відбувається на етапі написання програмного коду і дає можливість передбачити відсоток можливих збоїв у ПЗ комп'ютерної системи.

2.5. Висновки до розділу

У даному розділі одержано наступні наукові та практичні результати:

1. Проаналізовано підходи і стратегії тестування програмного забезпечення і встановлено, що важливим з точки зору підвищення продуктивності провадження процесу тестування є імплементація методу класифікації вимог та прогнозування дефектів коду при плануванні і формуванні тестових випадків на ранніх стадіях життєвого циклу .

2. Запропоновано метод класифікації вимог до програмного забезпечення, що базується на методах машинного навчання і використовує чотири алгоритми: метод опорних векторів, наївний байєсовий класифікатор, логістична регресія та метод найближчих сусідів, а також техніки опрацювання тексту: «мішок слів», TF-IDF і χ^2 , що в комплексі дало змогу забезпечити ефективність процесу класифікації вимог за групами функціональних і нефункціональних вимог, а нефункціональних в свою чергу за 11 підкласами вимог якості.

3. Запропоновано метод прогнозування дефектів програмного забезпечення у комп'ютерних системах, що використовує 6 алгоритмів машинного навчання і три набори дефектів і дає змогу забезпечити та передбачити з високою імовірністю можливість появи помилок у програмних модулях з врахуванням метрик програмного коду.

4. Обгрунтовано застосування метрик для оцінювання ефективності моделей та алгоритмів машинного навчання при проведенні класифікації вимог і прогнозуванні дефектів програмного забезпечення, що дає змогу обґрунтувати доцільність їхнього застосування при проведенні тестування.

РОЗДІЛ 3
АПРОБАЦІЯ МЕТОДІВ МАШИННОГО НАВЧАННЯ ПРИ КЛАСИФІКАЦІЇ ВИМОГ ТА ПРОГНОЗУВАННІ ДЕФЕКТІВ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ КОМП'ЮТЕРНИХ СИСТЕМ

3.1. Аналіз структури та даних при проведенні класифікації вимог до ПЗ

Для проведення експерименту щодо програмної реалізації методу класифікації вимог використано мову програмування Python і базу даних PROMISE_exp [13]. Цей репозиторій є розширеною версією оригінального набору даних PROMISE. Він є загальнодоступним і створеним на основі набору даних машинного навчання UCI, орієнтованим на перевірку та покращення моделей прогнозування у сфері програмної інженерії [19].

Оригінальний репозиторій складається з попередньо розміченого набору, який містить 255 функціональних вимог і 370 нефункціональних. Нефункціональні вимоги включають 11 різних типів. Розширена версія даного репозиторію включає 969 вимог розподіл за кількість вимог і класів якого представлено у табл. 3.1.

Таблиця 3.1

Кількість вимог за класами

Тип вимоги	Кількість
Функціональні (FR)	444
Доступність (A)	31
Сертифікати та ліцензії (L)	15
Естетичне оформлення (LF)	49
Ремонтопридатність (MN)	24
Оперативність (O)	77
Продуктивність (PE)	67
Масштабованість (SC)	22

Продовження табл. 3.1

Тип вимоги	Кількість
Безпека (SE)	125
Зручність використання (US)	85
Відмовостійкість (FT)	18
Портативність (PO)	12
Загальна кількість	969

Розширення відбулося за допомогою цілеспрямованого веб-пошуку документів, що містять записи вимог до ПЗ. Після цього фахівцями з даних та аналітиками було проведено аналіз документів, визначено і добуто відповідні вимоги.

Увесь процес визначав пріоритет сумісності та якості розширення, а результат розширення оцінювався за допомогою алгоритмів ML. Його результати порівнювалися з результатами вихідної бази даних, коли вони аналізувались за тими ж алгоритмами [15].

На рис. 3.1 наочно показано, наскільки незбалансованими є класи вимог. Єдиний випадок, коли класи добре розподілені, це коли всі підкласи нефункціональних вимог згруповані в один унікальний клас під час проведення бінарної класифікації. Таке розділення даних призводить до 444 функціональних і 525 нефункціональних вимог.

Усі документи базі даних пройшли процес нормалізації, який показаний у розділі 2 на рис. 2.1 – етапі 1 «Нормалізація».

На рис. 3.2 наведено скрін-шот вимог бази даних перед виконанням процесу нормалізації (очищення тексту).

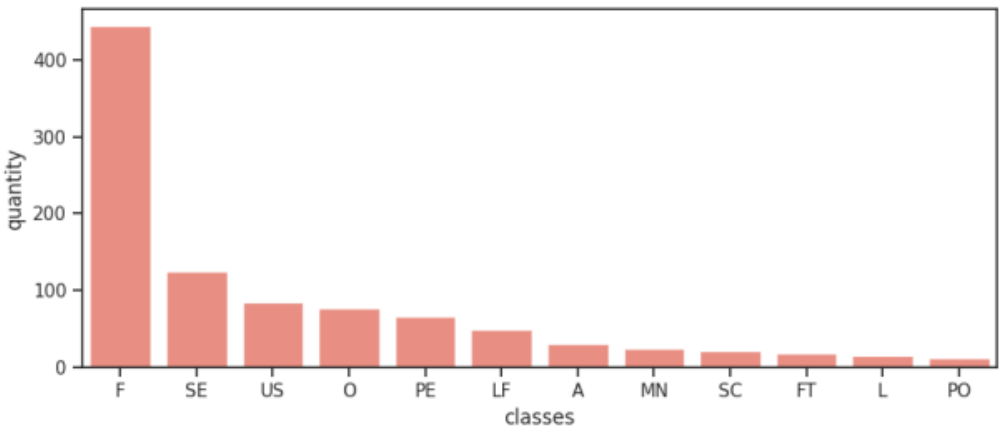

Рис. 3.1. Розподіл вимог за класами у фреймі даних

	Text	Class
1	The system shall refresh the display every 60 s.	PE
2	The application shall match the color of the schema set forth by Department of Homeland Security.	LF
3	If projected the data must be readable. On a 10 × 10 projection screen 90% of viewers must be able to read Event/Activity data from a viewing distance of 30.	US
4	The product shall be available during normal business hours. As long as the user has access to the client PC the system will be available 99% of the time during the first six months of operation.	A
5	If projected the data must be understandable. On a 10 × 10 projection screen 90% of viewers must be able to determine that Events or Activities are occurring in current time from a viewing distance of 100.	US
…	…	…
965	The system should be portable to various operating environments.	PO
966	Registered User must be able to maintain his/her session information for at least 60 min of inactive session before the system prompts him to log out of the system. The registered user must be provided with all the options of the E-store regardless of the time when he/she logs in.	F
967	The entire website must be user-friendly and easily navigable. The website must be provided with a site map for quick access to a particular link according to the requirement specification. The user must be able to find what he/she wants from the site without any difficulty. The website must adhere to branding schemes and the layout of the web pages must be uniform throughout.	US
968	The system shall support up to 10,000 simultaneous users against the central database at any given time and up to 5000 simultaneous users against the local servers at any one time. The performance of the website must be optimal increase of huge loads and hence appropriate load balancing must be done to achieve this. There can be any number of mirror servers readily available in case of huge loads without the user getting any delay.	PE
969	The website must provide highest degree of security to the registered users. All the transactions that are made must be secured. The sensitive information passed to and from the website must be secured. Identity theft and other security related issues must be solved. Unauthorized transmission of sensitive information of the user to third party websites for reference must be avoided. On the basis of user agreement the information must be processed. All the information about the registered user must be securely stored in the central database.	SE

Рис. 3.2. Знімок екрану з вимогами перед проведенням нормалізації тексту

3.2. Препроцесинг та векторизація тексту вимог до ПЗ

Процес нормалізації виконано із застосуванням бібліотеки NLTK, що дозволяє виконувати маніпуляції над природною мовою. Під час нормалізації всі слова у вимогах приведено до нижнього регістру. Наприклад, вимогу (таблиця 3.1,

ID=1): «Система повинна оновлювати дисплей кожні 60 секунд» було змінено на: «система повинна оновлювати дисплей кожну секунду», як показано у табл.3.2 (ID=1).

Після цього, слова, які відіграють не важливу роль або взагалі не мають значення вилучено з документів. Такі слова називаються стоп-словами і зазвичай є найпоширенішими, коли виконується об'єднання кількох документів. Такі слова, як «the», «a» тощо, вважаються нерелевантними до слова (табл. 3.2).

	Text	Class
1	System shall refresh display every second.	PE
2	Application shall match color schema set forth department homeland security.	LF
3	Project data must readable projection screen viewer must able read event activity data view distance.	US
4	Product shall available normal business hour long user access client pc system available time first six month operation.	A
5	Project data must understandable projection screen viewer must able determine event activity occur current time view distance.	US
...	...	...
965	System portable various operate environment.	PO
966	Register user must able maintain session information least minute inactive session system prompt log system registered user must provide option regardless time log.	F
967	Entire website must easily navigable website must provide site map quick access particular link accord requirement specification user must able find want site without difficulty website must adhere brand scheme layout web page must uniform throughout.	US
968	System shall support simultaneous user central database give time simultaneous user local server one time performance website must optimal in case huge load hence appropriate load balancing must do achieve number mirror server readily available case huge load without user get delay.	PE
969	Website must provide high degree security registered user transaction make must secure sensitive information pass website must secure identity theft security relate issue must solve unauthorized transmission sensitive information user third party website reference must avoid basis user agreement information must process information registered user must securely store central database.	SE

Рис. 3.2. Нормалізований текст вимог у наборі даних

Останнім кроком у процесі нормалізації було приведення слів до їх кореневої форми, наприклад, слово «користувачі» було змінено на «користувач», а «вказаний» — на «вказати». Код, який використовується для цього кроку, показаний у лістингу 3.1.

Лістинг 3.1. Приведення слів вимог до кореневої форми

```
#1 Remove blank rows .
Corpus [ 2 'text ']. dropna ( inplace = True )
#3 Change all the text to lower case .
Corpus [ 4 'text '] = [ entry . lower () for entry in Corpus
['text ']]
#5 Tokenization
Corpus [ 6 'text ']= [ word_tokenize ( entry ) for entry in
Corpus ['text ']]
#7 Remove Stop words , Non - Numeric and perfom Word
Lemmenting .
tag_map = defaultdict ( 8 lambda : wn. NOUN )
tag_map [ 9 'J'] = wn.ADJ
tag_map [ 10 'V'] = wn. VERB
tag_map [ 11 'R'] = wn.ADV
12
for 13 index , entry in enumerate ( Corpus ['text ']):
14
Final_words = [] 15
word_Lemmatized = WordNetLemmatizer () 16
for 17 word , tag in pos_tag ( entry ):
#18 Below condition is to check for Stop words and consider
only
alphabets 19 if word not in stopwords . words ('english ')
and 20 word . isalpha ():
word_Final = word_Lemmatized . lemmatize (word , tag_map [ tag
[0]]) 21
Final_words . append ( word_Final ) 22
Final_words = 23 ' '. join ( Final_words )
Corpus .loc [index , 24 'text '] = str ( Final_words )
Listing 1: Corpus leaning.
```

Після виконання нормалізації всі документи дата фрейму пройшли процес векторизації. Цей етап необхідний, щоб можна було використовувати інформацію,

добуту з документів у моделях машинного навчання, відповідно до етапу 2 «Вилучення ознак» (рис. 2.1).

Таким чином, використано BoW і TF-IDF в процесі векторизації. Дві форми виділення ознак (BoW і TF-IDF) порівнювалися на етапі класифікації, де можна спостерігати, які з цих методів покращили продуктивність використаних алгоритмів.

Щоб оцінити, які слова вважалися найбільш впливовими, обчислено суму рядків кожної матриці, отриману в результаті векторизації.

У табл. 3.3 показано 10 слів з найвищою частотою вживання для кожної техніки.

Таблиця 3.3

Частотні слова у фреймі даних

10 найбільш вживаних слів	Bag of Words	TF-IDF
1.	«must»	«must»
2.	«information»	«information»
3.	«user»	«sensitive»
4.	«website»	«website»
5.	«security»	«registered»
6.	«registered»	«security»
7.	«secure»	«secure»
8.	«sensitive»	«user»
9.	«high»	«third»
10.	«agreement»	«issue»

У табл. 3.3 можна помітити, що слова колонок за двома техніками збігаються, але з 3-ї позиції порядок важливості змінюється з BoW на TF-IDF.

Фаза 3 – це етап вибору ознак, де функції, створені в результаті їх добування, проходять процес фільтрації, щоб видалити деякі функції, які не є настільки важливими відповідно до якогось статистичного підходу, у даному випадку χ^2.

При застосуванні векторизації BoW і TF-IDF фільтрація ознак досягається за допомогою аргументу, використовуваного в обох методах вилучення ознак, «max_df», який використовується для ігнорування термінів, частота документів яких строго вище заданого порогу.

Іншим аргументом був «min_df», який ігнорує терміни, частота документів яких строго нижча, ніж заданий поріг, і значення цього аргументу також було досягнуто за його допомогою.

У підході χ^2 використано інший фільтр для функцій, який наведено у розділі 2, і одержаний за допомогою TF-IDF. Для цього використана функція «SelectKBest».

Ця функція приймає на вхід два аргументи, «score_func», тобто обрана функція оцінки (χ^2) і k, тобто кількості ознак з найкращим значенням χ^2, які будуть розглянуті. Щоб знайти значення k, застосовано той самий підхід оптимізації гіперпараметрів, який перед тим використано для налаштування BoW і TF-IDF.

Нормований та векторизований корпус використовувався для навчання та тестування продуктивності за допомогою чотирьох алгоритмів: kNN, SVM, MNB та LR (фаза 4 на рис. 2.1).

Ці алгоритми дають змогу провести класифікацію вимог до ПЗ за трьома різними типами деталізації:

– функціональні вимоги;
– нефункціональні вимоги;
– підкласи нефункціональних вимог.

В алгоритмах на основі методу опорних векторів, мультономіального байєсівського класифікатора та логістичної регресії застосовується гіперпараметр під назвою «class_weight». Даний гіперпараметр використовує значення мітки класів, щоб автоматично коригувати ваги, обернено пропорційні частотам класів у вхідних даних.

Усі гіперпараметри, як з алгоритмів векторизації, так і з алгоритмів класифікації, були обрані за допомогою функції під назвою «GridSearchCV». Ця функція виконує повний пошук за заданими значеннями параметра для оцінювача і вибирає найкращу комбінацію на основі деякого параметра оцінки, у даному випадку це F-міра.

Іншими словами, ця функція перевіряє всі можливі комбінації між параметрами і повертає комбінацію, яка досягла найкращого результату. Ця виграшна комбінація є комбінацією, яка використовується в процесі векторизації/класифікації.

Кількісне обчислення та оцінювання продуктивності чотирьох алгоритмів забезпечено за допомогою перехресної перевірки (крос-валідації) у відповідності до етапу 5 «Вимірювання ефективності» процесу класифікації, представленого на рис. 2.1.

Крос-валідація створює послідовність дій, де кожна підмножина даних використовується як навчальний набір, так і як набір для перевірки, обрано цей підхід для боротьби з незбалансованістю характеристик даних.

При проведенні експерименту, корпус був розділений на десять підмножин (10 разів), з яких 9 використовувалися для навчання алгоритмів, що відповідає 90 відсоткам бази, а 1 використовувався для виконання тестів, що відповідає 10 відсоткам бази даних.

На основі перехресної перевірки обчислено точність, повноту та F-міру результатів.

У лістингу 3.2 є функція, створена для виконання крос-валідації. У ній в якості параметрів передається:

– корпус (дані);
– алгоритм класифікації (модель);
– об'єкт, що відповідає за векторізацію корпусу (векторизатор);
– кількість крос-валідацій (у даному випадку їх 10).

Таким чином, функція добуває ознаки, навчає і тестує дані, відображаючи в кінці показники продуктивності, отримані за допомогою тесту.

Лістинг 3.2. Функція крос-валідації

```
def 1 kfoldcv (data , classifier , vectorizer , k_best ,
k = 10):
2kf = KFold ( n_splits =k, shuffle = True , random_state
= 60) 3
kf. get_n_splits ( data )
4report_test = pd. Series () 5
report_prediction = pd. Series ()
6 7
for train_indices , test_indices in kf. split ( data ):
train_text = data . iloc [ train_indices ][ 8 'text ']
train_class = data . iloc [ train_indices ][ 9 'class ']
10
test_text = data . iloc [ test_indices ][ 11 'text ']
test_class = data . iloc [ test_indices ][ 12 'class ']
13
pipeline = Pipeline ([( 14 'vect ', vectorizer ),
(15'chi ', SelectKBest ( score_func =chi2 , k= kbest )),
(16'clf ', classifier )])
17
model = pipeline . fit ( train_text , train_class ) 18
19
predictions = model . predict ( test_text ) 20
21
report_test = np. concatenate (( report_test , test_class
), axis =0) 22
report_prediction = np. concatenate (( report_prediction
, predictions ), axis =0) 23
24
print 25 ( metrics . classification_report ( report_test
, report_prediction , digits = 2))
```

Для підтримки експериментів використовувався інструмент Scikit-learn . Це модуль Python, який інтегрує кілька алгоритмів машинного навчання.

Цей інструмент був обраний, оскільки він містить алгоритми, використані в даному дослідженні, а також додаткові інструменти, які реалізують BoW, TF-IDF і перехресну перевірку (крос-валідацію). Крім того, в роботі виконано дослідження відмінностей використання BoW і TF-IDF при класифікації вимог.

3.3. Оцінювання ефективності алгоритмів машинного навчання при класифікації вимог до ПЗ

Для оцінювання ефективності реалізованих алгоритмів та моделі класифікації вимог до ПЗ на етапі тестування життєвого циклу використано три метрики щодо:

– ефективності бінарної класифікації вимог;

– ефективності при мультиноміальній (багатокласовій) класифікації нефункціональних вимог;

– ефективності при мультиноміальній класифікації вимог, що включає нефункціональні вимоги та функціональні вимоги.

Їхнє застосування дає змогу визначити найкращу комбінацію алгоритму машинного навчання та методики виділення ознак для класифікації вимог.

У контексті цих трьох експериментів використовували алгоритми машинного навчання (ML), описані у розділі 2.

Для оцінки алгоритмів використовувалися такі показники:

– метрика точності;

– метрика повноти;

– F-міра.

Оцінку ефективності алгоритмів машинного навчання при бінарній класифікації вимог до ПЗ проведено з метою визначення якості розрізнення функціональних вимог та нефункціональних вимог.

У табл. 3.4 представлені результати, отримані при бінарній класифікації. Можна помітити, що метод опорних векторів і логістична регресія володіють

найкращими показники при класифікації за показниками продуктивності при значенні 0,91 за допомогою TF-IDF.

Таблиця 3.4

Показники ефективності алгоритмів бінарної класифікації

			Алгоритм машинного навчання			
			Метод опорних векторів	Байєсовий класифікатор	Метод найближчих сусідів	Логістична регресія
Техніки векторизації тексту	«Мішок слів» (BoW)	Точність	0,90	0,91	0,82	0,88
		Повнота	0,90	0,91	0,82	0,88
		F1	0,90	0,91	0,82	0,88
	TF-IDF	Точність	0,91	0,91	0,87	0,91
		Повнота	0,91	0,90	0,87	0,91
		F1	0,91	0,90	0,87	0,91
	χ^2	Точність	0,90	0,89	0,84	0,89
		Повнота	0,90	0,89	0,84	0,89
		F1	0,90	0,89	0,84	0,89

Алгоритм SVM також був алгоритмом, який показав найменші відхилення при оцінюванні його продуктивності, що становить більше або рівна 0,90 за всіма показниками.

Алгоритм мультономіального байєсового класифікатора також мав чудову продуктивність з TF-IDF. Однак з різницею в 0,01 F-міра методу опорних векторів та логістичної регресії була вищою.

За цими результатами можна зробити висновок, що алгоритми на основі методу опорних векторів та логістичної регресії показали найкращі результати при бінарній класифікації вимог до ПЗ.

У табл. 3.5 представлені результати оцінки алгоритмів ML, які використовуються для розв’язання задачі мультиноміальної класифікації нефункціональних вимог.

Таблиця 3.5

Показники ефективності алгоритмів мультиноміальної класифікації

			Алгоритм машинного навчання			
			Метод опорних векторів	Байєсовий класифікатор	Метод найближчих сусідів	Логістична регресія
Техніки векторизації тексту	«Мішок слів» (BoW)	Точність	0,68	0,71	0,56	0,71
		Повнота	0,67	0,73	0,48	0,71
		F1	0,66	0,72	0,49	0,70
	TF-IDF	Точність	0,73	0,71	0,66	0,75
		Повнота	0,73	0,71	0,66	0,75
		F1	0,72	0,71	0,66	0,74
	χ^2	Точність	0,72	0,69	0,62	0,70
		Повнота	0,71	0,70	0,63	0,71
		F1	0,71	0,68	0,62	0,70

У цьому експерименті оцінено ефективність класифікації нефункціональних вимог, що належать до 11 різних класах, відповідно до міток, початково визначених у базі даних PROMISE_exp.

У цій класифікації відбулося значне зменшення кількості екземплярів, оскільки база даних має 444 функціональні вимоги, то класифікація 11 класів нефункціональних вимог має лише 525 екземплярів.

Порівняно з іншими аспектами, класифікатори мали більшу складність у диференціації функціональних вимог. Продуктивність алгоритмів була гіршою в

порівнянні з іншими, в основному за алгоритмом kNN у поєднанні з Bow, з повнотою 0,48.

Однак відбулося покращення продуктивності алгоритмів при використанні TF-IDF. За результатами можна визначити, що логістична регресія у поєднанні з TF-IDF перевершує по ефективності метод опорних векторів, мультиноміальний байєсовий класифікатор та метод найближчих сусідів.

У табл. 3.6 представлено результати експерименту, одержані за допомогою трьох методів виділення ознак «мішок слів», TF-IDF і χ^2 у поєднанні з чотирма алгоритмами машинного навчання, які показують середньозважену точність, повноту та F-метрику результатів класифікації за 12 різними показниками – функціональні вимоги та 11 типів нефункціональних вимог. Загалом, точність, повнота та F-міра всіх комбінацій техінік і алгоритмів є більшою або рівною 0,60.

Таблиця 3.6

Показники ефективності алгоритмів за 12 мітками класів

			Алгоритм машинного навчання			
			Метод опорних векторів	Байєсовий класифікатор	Метод найближчих сусідів	Логістична регресія
Техніки векторизації тексту	«Мішок слів» (BoW)	Точність	0,73	0,77	0,63	0,76
		Повнота	0,72	0,77	0,60	0,77
		F1	0,72	0,77	0,60	0,76
	TF-IDF	Точність	0,78	0,76	0,72	0,78
		Повнота	0,78	0,77	0,73	0,79
		F1	0,78	0,76	0,72	0,78
	χ^2	Точність	0,77	0,74	0,67	0,7
		Повнота	0,77	0,74	0,69	0,77
		F1	0,76	0,73	0,68	0,76

Алгоритм логістичної регресії досяг найкращої класифікації за продуктивністю зі значенням показника повноти на рівні 0,79. Найбільш значуща різниця в продуктивності була в мірі повноти k-найближчих сусідів зі збільшенням на 21,67% від BoW до TF-IDF. У третьому випадку зернистості логістична регресія була найкращим алгоритмом.

Щоб оцінити ефективність класифікації, одержану комбінацією TF-IDF і методу опорних векторів, проаналізовано відповідні показники продуктивності, одержані за допомогою алгоритму для кожної з міток у базі даних, як показано у табл. 3.7.

Таблиця 3.7

Функціонали якості при застосуванні методу опорних векторів та TF-IDF для 12 класів вимог до ПЗ

Тип вимоги	Точність	Повнота	F1-метрика
Функціональність	0,87	0,92	0,89
Доступність	0,77	0,74	0,75
Сертифікати та ліцензії	0,88	0,47	0,61
Естетичне оформлення	0,78	0,65	0,71
Ремонтопридатність	0,52	0,50	0,51
Оперативність	0,64	0,53	0,58
Продуктивність	0,82	0,82	0,82
Масштабованість	0,57	0,55	0,56
Безпека	0,71	0,77	0,74
Зручність використання	0,66	0,72	0,69
Відмовостійкість	0,70	0,39	0,50
Портативність	0,50	0,17	0,25

Варто зазначити, що для функціональних вимог найвищою є F1-міра через велику кількість спостережень. Також можна спостерігати хорошу класифікацію

вимог щодо вимог продуктивності – 0,82 для всіх показників, що є вимогою з другим найбільшим показником за F1-мірою і навіть не є класом із другою більшою кількістю спостережень.

Найгіршою класифікацією були вимоги до переносимості (портативність) через найменшу кількість екземплярів цих вимог у PROMISE_exp.

Іншим релевантним спостереженням є ефективність класифікації вимог безпеки, яка, незважаючи на велику кількість екземплярів (125 екземплярів), отримала низьку точність щодо її повноти та оцінки F1-метрики. Це відбулося через те, що класифікація стала залежною від зовнішнього вигляду ключових слів, таких як «пароль» і «файли cookie».

Для кращої візуалізації та розуміння на рис. 3.3 показано співвідношення між часткою кожного типу вимог та отриманою F-мірою.

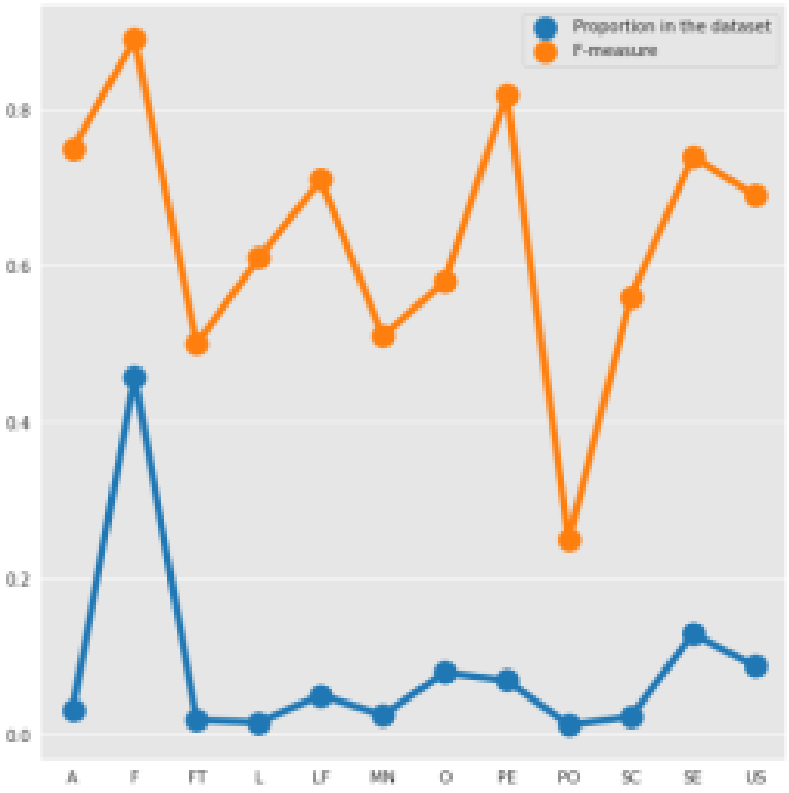

Рис. 3.3. Відношення між розподілом вимог та F1-мірою

Можна помітити, що існує взаємозв’язок серед цих двох мір, тому що їхні графіки мають подібну форму. Як видно з рис. 3.3, для низьких значень частки

незначного збільшення чи великого зменшення кількості екземплярів має значний вплив на F-міру. Більшість значень F-міри збільшується, коли частка збільшується, але на відміну від інших вимог, F-міра продуктивності збільшується у порівнянні з оперативністю, коли кількість екземплярів зменшується.

Крім того, хоча відмінності пропорцій функціональних вимог до вимог продуктивності, вимог доступності до вимог безпеки дуже відрізняються, F-міри цих кортежів є дуже схожими.

3.4. Алгоритм і результати експериментальних досліджень при прогнозуванні дефектів ПЗ

У цьому експерименті було проаналізовано дані з трьох наборів, що містить інформацію про дефекти, яка використовується для побудови моделі їхнього прогнозування.

Після аналізу структури дата фреймів, виконувався попередній препроцесинг даних. Коваріаційний та кореляційний аналіз дав змогу визначити високу кореляцію всередині даних, відновити відсутні значення, добути ознаки, провести нормалізацію, знайти мінімальні та максимальні значення, застосувати методи матричної декомпозиції PCA для зменшення розмірності розріджених матриць. тощо. Алгоритм виконання дій при побудові моделі та імплементації алгоритмів машинного навчання показано на рис. 3.4.

У цьому дослідженні основна увага зосереджена на автоматизованому відновленні несправностей всередині програмного забезпечення за допомогою прогнозної моделі. Крім того, це дозволяє моніторити як дефектні, так і несправні класи.

В експерименті використано 10-кратну техніку перехресної перевірки, щоб оцінити ефективність шести методів класифікації. Для визначення параметрів моделі програмного дефекту застосовано різні методи попередньої обробки даних, які підвищили точність і узгодженість моделі класифікації.

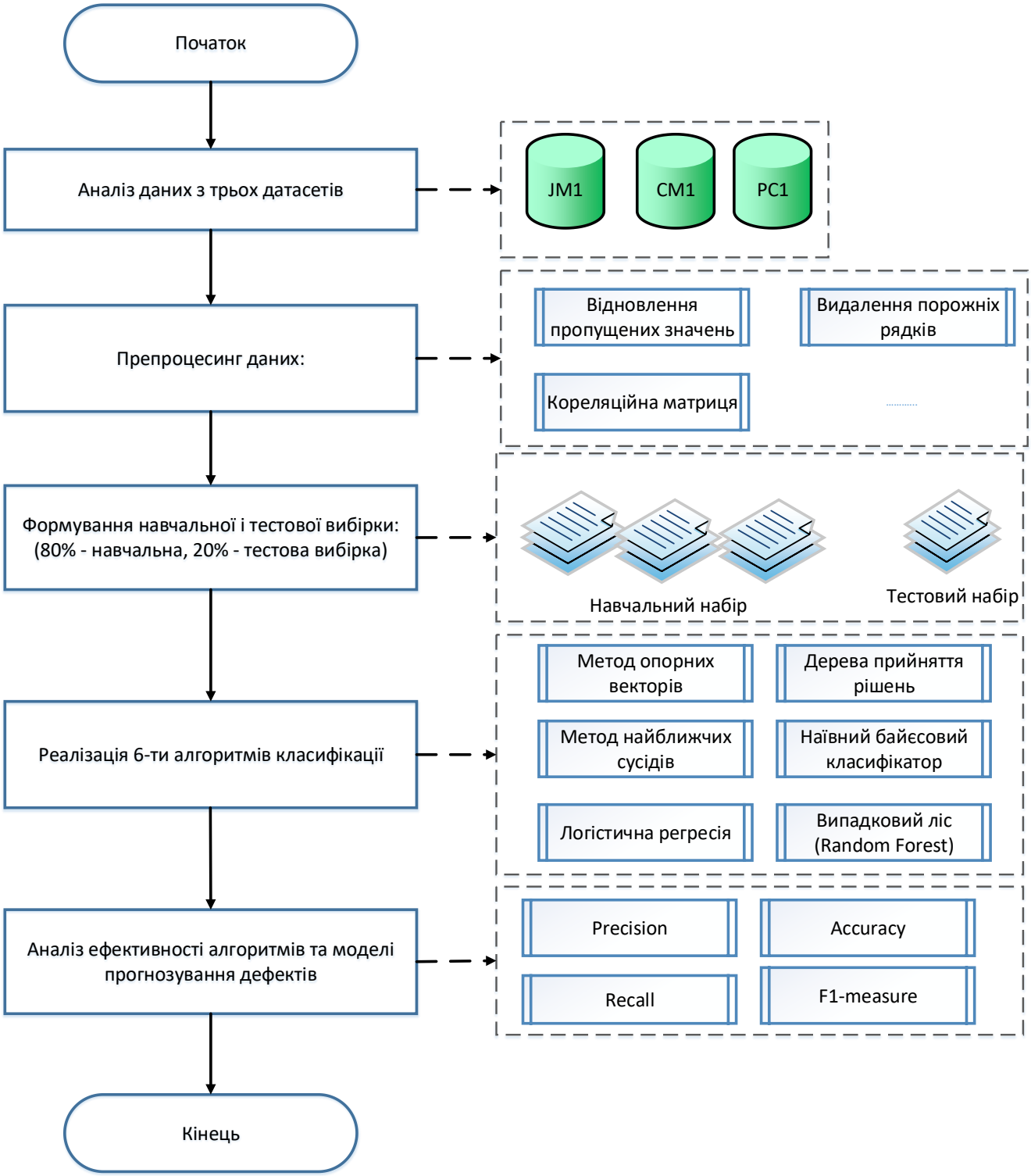

Рис. 3.4. Алгоритм побудови моделі та алгоритмів прогнозування дефектів ПЗ

Табл. 3.8 показує оцінку продуктивності шести методів класифікації для прогнозування помилок у ПЗ. Що стосується точності, то:

– алгоритм на основі дерев прийняття рішень та метод опорних векторів досягли найвищої продуктивності на наборах даних JM1.

– алгоритми дерева прийняття рішень, наївний байєсівський класифікатор, метод опорних векторів та випадковий ліс показали найкращу продуктивність на наборах даних CM1;

– алгоритми дерев прийняття рішень, метод опорних векторів і випадковий ліс отримали найвищу продуктивність (тобто 97%) на наборах даних PC1.

Таблиця 3.8

Ефективність алгоритмів машинного навчання при прогнозуванні дефектів ПЗ

Алгоритм	Функціонал якості	Набір даних		
		JM1	CM1	PC1
Дерево прийняття рішень	Точність (Precision)	0,99	0,99	0,97
	Точність (Accuracy)	0,98	0,97	0,98
	Повнота (Recall)	0,98	0,99	0,99
	F1-метрика	0,98	0,99	0,97
Наївний байєсівський класифікатор	Точність (Precision)	0,93	0,99	0,90
	Точність (Accuracy)	0,97	0,99	0,97
	Повнота (Recall)	0,92	0,99	0,84
	F1-метрика	0,92	0,99	0,86
Метод опорних векторів	Точність (Precision)	0,99	0,99	0,96
	Точність (Accuracy)	0,98	0,99	0,98
	Повнота (Recall)	0,99	0,99	0,99
	F1-метрика	0,99	0,99	0,97
Логістична регресія	Точність (Precision)	0,94	0,95	0,94
	Точність (Accuracy)	0,98	0,98	0,97
	Повнота (Recall)	0,96	0,95	0,92
	F1-метрика	0,94	0,95	0,92

Продовження табл. 3.8

Алгоритм	Функціонал якості	Набір даних		
		JM1	CM1	PC1
Випадковий ліс	Точність (Precision)	0,98	0,99	0,97
	Точність (Accuracy)	0,98	0,99	0,98
	Повнота (Recall)	0,99	0,99	0,96
	F1-метрика	0,98	0,99	0,97
Метод найближчих сусідів	Точність (Precision)	0,95	0,95	0,86
	Точність (Accuracy)	0,99	0,99	0,95
	Повнота (Recall)	0.96	0.99	0.88
	F1-метрика	0.95	0.97	0.86

Відносно всі класифікатори показали хороші показники з точки зору точності. Однак, враховуючи повноту аналізу, метод опорних векторів і випадковий ліс досягли найвищої продуктивності на наборах даних JM1. Логістична регресія і наївний байєсівський класифікатор досягли найнижчої продуктивності на наборах даних CM1 і PC1. Ще одна міра оцінювання якості класифікації – це міра F1.

Щодо показника F1, то метод опорних векторів досяг найвищого значення на наборах даних JM1, а NB отримав найнижчий бал (93%). Переглядаючи набори даних CM1, ми можемо відстежити, що оцінки F1 здебільшого схожі (тобто наївний байєсовий класифікатор, дерева прийняття рішень, метод опорних векторів, випадковий ліс мають ефективність 99%, а метод найближчих сусідів 97%, логістична регресія – 95%).

Крім того, випадковий ліс забезпечує найкращий результат (тобто 99%), а KNN – найнижчий (86%) на наборах даних PC1. Всі класифікатори забезпечили максимальну ефективність в наборах даних JM1, CM1 і PC1 з точки зору точності. Це вказує на те, що вони дуже ефективні для прогнозування модулів з дефектами у програмному забезпеченні.

Таким чином, запропоновано метод автоматизованого підходу для інженерії програмного забезпечення на етапі тестування програмного коду комп’ютетрної системи, що дає змогу з високою точністю прогнозувати дефекти.

Після цього головною метою нашого дослідження було оцінити здібності шести методів класифікації, заснованих на нагляді, передбачити модулі дефектів програмного забезпечення з використанням 3 наборів даних NASA. Результати (тобто точність: 98-99%) експерименту з різними атрибутами показали здатність та ефективність моделі виявляти несправність та покращувати якість програмного забезпечення.

Крім того, ця модель прогнозування може бути застосовано при ранньому виявленні збоїв програмного забезпечення, збираючи дані розробки програмного забезпечення в реальному часі з цільових програм.

Запропонований підхід може бути використаний для відновлення працездатності ПЗ після збоїв всередині системи та покращити життєвий цикл комп’ютерної системи шляхом імплементації методів машинного навчання для більш ефективного пошуку помилок у ПЗ.

3.5. Висновки до розділу

1. Проведено аналіз набору даних, виконано нормалізацію та векторизацію тексту вимог до програмного забезпечення, а також засобами мови Python та за допомогою відкритих бібліотек машинного навчання реалізовано відповідні алгоритми щодо класифікації вимог до ПЗ, які дають змогу забезпечити точність на рівні класифікації при бінарній класифікації на рівні 90%, а мультиноміальній – 70%.

2. Програмно імплементовано метод і алгоритм прогнозування дефектів програмного забезпечення, що дало змогу обґрунтувати доцільність його використання на етапі тестування програмного забезпечення комп’ютерних систем, оскільки точність прогнозування, у більшості випадків, досягає рівня 98-99% .

РОЗДІЛ 4
ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

4.1. Охорона праці

Тема кваліфікаційної роботи магістра пов’язана із дослідженням методів і засобів тестування програмного забезпечення комп’ютерних систем з використанням алгоритмів машинного навчання. Такі роботи передбачають використання комп’ютерної техніки на етапах формування пояснювальної записки, налаштування засобів автоматизації процесу управління тестуванням з використанням алгоритмів машинного навчання. Тому, при виконанні робіт з тестування програмних складових комп’ютерних систем необхідно враховувати вимоги з охорони праці при експлуатації комп’ютерної техніки.

В Україні діють ряд законів, нормативних документів та актів, які регулюють процеси забезпечення та управління охороною праці у різних галузях народного господарства. До них належать: Конституція України, Закони України "Про охорону праці", "Про охорону здоров'я", "Про пожежну безпеку", "Про забезпечення санітарного та епідемічного благополуччя населення", "Про загальнообов'язкове державне соціальне страхування від нещасного випадку на виробництві та професійного захворювання, які спричинили втрату працездатності", Кодекс законів про працю України (КЗпП).

Однією з основних вимог до приміщень, де робочі місця обладнані комп’ютерною технікою і планується використання програмного комплексу для забезпечення процесу тестування ПЗ, є вимоги щодо площі, яка відводиться на один ПК. При проектуванні автоматизованих робочих місць тестувальників програмного забезпечення необхідно дотримуватись вимог щодо розміщення комп’ютерів. На один ПК передбачено площу 6 $м^2$ та об’єм 20 $м^3$.

Однак робота з комп’ютером включає різні завдання, які об'єднуються такими загальними чинниками, як те, що робота проводиться в сидячому положенні і вимагає уважного, неперервного та іноді тривалого спостереження.

Перше правило, якого варто дотримуватись тестувальникам програмного забезпечення стосується правильного облаштування робочого столу. При цьому слід передбачити наступні його параметри: фіксована висота – 720 мм, забезпечення необхідного простору для рук по висоті, ширині і глибині, в області сидіння не повинно бути шухляд.

Друге правило визначає облаштування робочого стільця: можливість регулювання висоти стільця, забезпечення обертання конструкції стільця. У приміщеннях з ПК, на яких планується виконання задач з тестування програмних складових комп'ютерних систем, яскравість знаків і яскравість фону дисплею повинна бути спроектована таким чином, щоб не було великої відмінності з яскравістю навколишнього середовища, але знаки повинні чітко розпізнаватися на відстані читання. Характеристики освітлення, зокрема у приміщеннях, де експлуатується ПК, повинні відповідати ДБН В.2.5-28-2006 "Природнє і штучне освітлення". Основні вимоги даного нормативного документу стосуються забезпечення наступних вимог:

- освітлення з лівої сторони;
- рівномірне освітлення всього робочого простору;
- комп'ютерна техніка встановлюється у місцях, віддалених від вікон;
- встановлення непрямого штучного освітлення;
- світло, що поступає через вікна, «пом'якшують» за допомогою штор;
- робоче місце організовується так, щоб напрям погляду був паралельним фронту вікон.

Ще одне правило, якого слід дотримуватись тестувальникам програмного забезпечення, передбачає оптимальний метод роботи, що полягає у передбачені зміни завдань і навантажень, дотримання перерви в роботі: 5 хвилин через 1 годину роботи біля дисплея або 10 хвилин після 2-х годин роботи біля дисплея. Вимоги цього правила регламентовані нормативним документом «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин».

При створенні сприятливих умов для підвищення продуктивності і зменшення напруги значну роль грають чинники, що характеризують стан навколишнього середовища: мікроклімат приміщення, рівень шуму і освітлення.

Рекомендована величина відносної вологості, яка повинна бути забезпечена у приміщеннях з експлуатації програмного комплексу поведінкового тестування програмних складових комп'ютерних систем, повинна відповідати НПАОП 0.00-7.15-18 і становити 65 – 70%. При цьому робоче місце повинно бути добре вентильованим.

У даний час з погляду шумового навантаження досягнуто значного прогресу. Рівень шуму в приміщеннях (приблизно 40 Дб) не перевищує допустимого рівня, незалежно від кількості використовуваного обладнання. Для приміщень, в яких експлуатується програмний комплекс підтримки запропонованих методів поведінкового тестування, потрібно забезпечити виконання вимог пожежної безпеки, які визначені Правилами пожежної безпеки в Україні, НПАОП 0.00-7.15-18 «Вимоги щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями» [24].

Будівлі і ті їх частини, в яких розташовуються ПК можуть належати до II ступеня вогнестійкості. Над та під приміщеннями, де розташовуються ПК, а також у суміжних з ними приміщеннях не дозволяється розташування приміщень категорій А і Б за вибухопожежною небезпекою. Приміщення категорії В повинні бути відділеними від приміщень з ПК протипожежними стінами.

Таким, чином при дослідженні методів і засобів поведінкового тестування програмних складових комп'ютерних систем, встановлено, що найбільш повним нормативним документом щодо охорони праці користувачів ПК, до яких належать тестувальники програмного забезпечення, є НПАОП 0.00-7.15-18 «Вимоги щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями». Дотримання вимог, які неведені у цьому документі, сприяє зниженню негативного впливу ПК, його компонентів та інших зовнішніх пристроїв на тестувальників, які проводять роботи щодо перевірки правильності функціонування програмних складових комп'ютерних систем.

4.2. Підвищення стійкості роботи об'єктів господарської діяльності у воєнний час

На основі вивчення факторів, які впливають на стійкість роботи об'єктів господарської діяльності, та оцінки стійкості елементів і галузей виробництва проти уражаючих факторів ядерної, хімічної і біологічної зброї, стихійних лих і виробничих аварій, необхідно завчасно організувати і провести організаційні, інженерно-технічні й технологічні заходи для підвищення стійкості роботи.

Здійснення організаційних заходів передбачає завчасну підготовку всіх структур цивільного захисту, служб і формувань до надзвичайних ситуацій, в тому числі і військових дій.

Вжиттям технологічних заходів підвищується стійкість роботи об'єктів шляхом змінювання технологічних процесів, режимів, можливих в умовах різних надзвичайних ситуацій.

Інженерно-технічні заходи мають забезпечити підвищену стійкість виробничих споруд, технологічних ліній, устаткування, комунікацій об'єкта до впливу уражаючих факторів під час військових дій.

При проведенні цих заходів необхідно враховувати конкретні умови об'єкта народного господарства. Проте є загальні організаційні інженерно-технічні заходи, які мають проводитись на всіх об'єктах.

Одним з найбільш важливих завдань в умовах воєнного часу і надзвичайних ситуацій є забезпечення захисту людей та їх життєдіяльності.

Для підвищення стійкості об’єктів господарювання та захисту людей необхідно:

– створити на об'єкті надійну систему оповіщення про загрози нападу противника, радіоактивне забруднення, хімічне і біологічне зараження, загрозу стихійного лиха і виробничої аварії.

– організувати розвідку і спостереження за радіоактивним забрудненням, хімічним і біологічним зараженням;

– організувати гідрометеорологічне спостереження за рівнем води, напрямком і швидкістю вітру, рухом і поширенням хмари радіоактивного забруднення, сильнодіючих отруйних речовин і отруйних речовин.

– створити фонд захисних споруд ЦО, запасів засобів індивідуального захисту і забезпечення своєчасної видачі їх населенню.

– завчасно підготуватись до масової санітарної обробки населення і знезаражування одягу;

– організувати взаємодію з установами охорони здоров'я для медичного обслуговування населення в умовах воєнного часу.

Також в умовах воєнного часу необхідно провести підготовку до евакуації населення, розміщеного в зонах можливих руйнувань і катастрофічного затоплення. Це передбачає завчасну підготовку місць евакуації, організацію прийому евакуйованого населення на територію населених пунктів.

Окрім цього, необхідно забезпечити постачання продуктів харчування, питної води, предметів першої необхідності та провести заходи щодо морально-психологічної підготовки населення до виживання в умовах воєнного часу, забезпечити процес чіткого інформування про обстановку та правила дій і поведінки населення в надзвичайних ситуаціях воєнного часу.

Для забезпечення стійкості роботи об'єктів повинні проводитись інженерно-технічні заходи на мережах комунального господарства з метою захисту джерел тепла із заглибленням у ґрунт комунікацій. Котельні слід розміщувати в спеціальному окремо розміщеному приміщенні.

Якщо об'єкт одержує тепло з міської теплоцентралі, необхідно провести заходи для забезпечення стійкості трубопроводів і розподільних пристроїв, підведених до об'єкта.

Теплова мережа має будуватися за кільцевою системою з прокладанням труб у спеціальних каналах зі з'єднанням паралельних ділянок. Для відключення пошкоджених ділянок мають бути встановлені запірно-регулюючі засувки, вентилі та ін. Ці пристосування необхідно розміщувати в оглядових колодязях, на території, що не завалюється при руйнуванні будівель.

Система каналізації має будуватись окремо: одна для дощових, друга для промислових і господарських вод. На об'єкті має бути не менше двох виводів з підключенням до міських каналізаційних колекторів, а також виводи і колодязі з аварійними засувками на об'єктових колекторах з інтервалом 50 м на території, що не завалюється, для аварійного скидання неочищеної води в найближчі штучні та природні заглиблення.

На деяких промислових об'єктах є системи для забезпечення технології виробництва: для подання кисню, аміаку, стиснутого повітря та інших рідких і газових реактивів. Для цих систем розробляють заходи для попередження виникнення вторинних факторів зброї, стихійних лих та виробничих аварій і катастроф.

Створення резерву енергетичних потужностей за рахунок автономних пересувних електростанцій, а також місцевих джерел електроенергії. Підготовка автономних електростанцій до роботи за спеціальним режимом (графіком) для забезпечення технологічних процесів виробництва, для яких неможливі тривалі перерви в електропостачанні.

З метою попередження аварій на електричних мережах необхідно установити автоматичну систему відключення при виникненні перенапруги. Повітряні лінії електропостачання замінити на підземно-кабельні.

Створення необхідних запасів (резервів) паливно-мастильних матеріалів та інших видів палива й організація їх безпечного зберігання.

Щоб не допустити зупинки підприємства через дефіцит палива, необхідно підготуватись для роботи на різних видах палива: нафта, вугілля, газ.

Для підвищення стійкості забезпечення водою слід провести такі заходи. Необхідно створити основні і резервні джерела водопостачання. Як резервне джерело краще мати артезіанську свердловину, яку необхідно підключити до системи водопостачання. Крім того, воду можна брати з близько розміщеної природної водойми або спорудити штучну водойму чи резервуари з обладнанням пристроїв для збору і перекачування води.

Всі ділянки водопостачання повинні бути заглиблені в ґрунт з обладнанням пожежних гідрантів і пристроїв для відключення пошкоджених ділянок. Локальні мережі водопостачання окремих великих підприємств варто з'єднати із загальноміською системою водопостачання в єдине кільце.

Підвищенню стійкості забезпечення водою сприяє подавання води безпосередньо в мережу поза водонапірними баштами, спорудження обвідних ліній для подання води поза пошкодженими спорудами.

Завчасне вжиття заходів захисту вододжерел, водопровідних споруд, свердловин і шахтних колодязів від забруднення радіоактивними речовинами, зараження хімічними і біологічними засобами.

Підготовка меліоративних, гідротехнічних та іригаційних споруд і систем до експлуатації в надзвичайних умовах.

Для забезпечення виробництва продукції необхідні електроенергія, паливо, мастила, засоби захисту рослин, міндобрива, профілактичні й лікувальні препарати ветеринарної медицини, запасні частини, сировина та інші матеріально-технічні засоби. Забезпечення об'єктів цими ресурсами дасть можливість випускати необхідну продукцію в надзвичайних умовах мирного і воєнного часу. Тому повинні проводитись такі заходи, які б забезпечили стійкість постачання і сприяли підвищенню захисту мережі електро-, водо-, газопостачання, транспортних комунікацій і джерел постачання всім необхідним для забезпечення функціонування галузей сільського господарства в надзвичайних умовах.

З метою попередження аварій на електричних мережах необхідно встановити автоматичну систему відключення перенапруги. Повітряні лінії електропостачання слід замінити на підземно-кабельні.

Газ використовується як паливо і на хімічних підприємствах у технологічному процесі. Для безперебійного забезпечення газом, газові мережі необхідно підводити до об'єкта з двох напрямків, які мають бути з'єднані в єдине кільце з обладнанням для можливого дистанційного автоматичного управління й у разі необхідності відключення пошкоджених ділянок.

На великих підприємствах необхідно мати підземні ємності із закачаним резервним газом.

На підприємствах, де використовується пара, необхідно захистити джерела його постачання, заглибити в ґрунт комунікації паропостачання і встановити запірні пристосування.

Запас резервних матеріалів необхідно розраховувати на такі строки роботи підприємства, за які можливе відновлення регулярного постачання.

Передбачити, на випадок перебоїв в постачанні підприємствами-суміжниками, створення місцевих матеріалів, сировини для виготовлення комплектуючих виробів і інструментів силами свого підприємства.

Для підвищення стійкості та забезпечення збереження (відновлення) будівель і споруд в умовах воєнного часу необхідно:

– провести оцінку можливих ступенів руйнування будівель і споруд господарства населеного пункту, визначити обсяг невідкладних ремонтних робіт, потреби в будівельних матеріалах.

– створити і підготувати спеціальні формування для ремонтно-відновних, будівельних та інших робіт на об'єкті.

– розробити комплекс протипожежних заходів, які виключали б можливість виникнення масових пожеж.

Для забезпечення надійності системи управління і зв'язку потрібно організувати захищений пункт управління, забезпечити його засобами зв'язку, які б дали можливість швидко доводити сигнали ЦЗ до всіх виробничих підрозділів і населення у місцях проживання. При цьому необхідно здійснити планування збору даних про обстановку, передачу команд і розпоряджень в умовах впливу на об'єкт уражаючих факторів. Для підвищення стійкості системи управління і зв’язку в умовах воєнного часу необхідно організувати використання радіозасобів, засобів телефонного зв'язку, а також забезпечити зв’язок із колонами евакуйованого населення, що перебувають у дорозі, і відповідальними особами, які супроводжують їх під час евакуації, забезпечити дублювання ліній і каналів зв'язку.

Висновок.

Отже, підвищення стійкості роботи об'єктів господарської діяльності у воєнний час можна забезпечити шляхом організації і проведення сукупності заходів, які включають організаційні, інженерно-технічні й технологічні заходи. Надзвичайно важливим є планування та організація захисту комплексів критичної інфраструктури, зокрема водопостачання, водовідведення, газопостачання та зв’язку.

ВИСНОВКИ

Основні наукові та практичні результати полягають в наступному.

1. Проведено аналіз важливих понять, принципів і послідовності виконання процесів, що використовуються при проектуванні комп'ютерних систем, зокрема, термінологічні особливості у процесі імплементації програмного забезпечення на етапі тестування, що дало змогу зрозуміти і в подальшому визначити шляхи імплементації методів машинного навчання для підвищення ефективності виконання стадій життєвого циклу.

2. Проаналізовано життєвий цикл помилок у програмному забезпеченні, що дало можливість врахувати особливості їх виявлення та усунення на етапі тестування з врахуванням підходів та методів розробки комп'ютерних систем та імплементацією алгоритмів машинного навчання щодо прогнозування дефектів у програмних модулях.

3. На основі аналізу і класифікації методів тестування визначено необхідність впровадження методу машинного навчання щодо класифікації вимог до програмного забезпечення, що в перспективі дозволить використовувати їх при створенні сценаріїв тестування та формуванні звітів з тестування.

4. Проаналізовано підходи і стратегії тестування програмного забезпечення і встановлено, що важливим з точки зору підвищення продуктивності провадження процесу тестування є імплементація методу класифікації вимог та прогнозування дефектів коду при плануванні і формуванні тестових випадків на ранніх стадіях життєвого циклу .

5. Запропоновано метод класифікації вимог до програмного забезпечення, що базується на методах машинного навчання і використовує чотири алгоритми: метод опорних векторів, наївний байєсовий класифікатор, логістична регресія та метод найближчих сусідів, а також техніки опрацювання тексту: «мішок слів», TF-IDF і χ^2 , що в комплексі дало змогу забезпечити ефективність процесу класифікації вимог за групами функціональних і нефункціональних вимог, а нефункціональних в свою чергу за 11 підкласами вимог якості.

6. Запропоновано метод прогнозування дефектів програмного забезпечення у комп’ютерних системах, що використовує 6 алгоритмів машинного навчання і три набори дефектів і дає змогу забезпечити та передбачити з високою імовірністю можливість появи помилок у програмних модулях з врахуванням метрик програмного коду.

7. Обґрунтовано застосування метрик для оцінювання ефективності моделей та алгоритмів машинного навчання при проведенні класифікації вимог і прогнозуванні дефектів програмного забезпечення, що дає змогу обґрунтувати доцільність їхнього застосування при проведенні тестування.

8. Проведено аналіз набору даних, виконано нормалізацію та векторизацію тексту вимог до програмного забезпечення, а також засобами мови Python та за допомогою відкритих бібліотек машинного навчання реалізовано відповідні алгоритми щодо класифікації вимог до ПЗ, які дають змогу забезпечити точність на рівні класифікації при бінарній класифікації на рівні 90%, а мультиноміальній – 70%.

9. Програмно імплементовані метод і алгоритм прогнозування дефектів програмного забезпечення, що дало змогу обґрунтувати доцільність його використання на етапі тестування програмного забезпечення комп’ютерних систем, оскільки точність прогнозування, у більшості випадків, досягає рівня 98-99% .

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Бейзер Б. Тестування чорної скриньки. Технології функціонального тестування програмного забезпечення і систем. Питер, 2014. 320 с.

2. Майерс Г. Искусство тестирования программ, 3-е издание = The Art of Software Testing, 3rd Edition М.: «Диалектика». 2012. 272 с.

3. Синицын С. В. Верификация программного обеспечения. М.: БИНОМ, 2008. 368 с

4. Черный и белый ящик. URL: http://b-test.narod.ru/black_and_white_box.htm (дата звернення 11.9.2021 р.).

5. Lutskiv A. Adaptable Text Corpus Development for Specific Linguistic Research / Andriy Lutskiv, Nataliya Popovych // International Scientific-Practical Conference «Problems of Infocommunications. Science and Technology» (October 8-11, 2019), 2019. - C.217-223.

6. Lutskiv A. Big Data Approach to Developing Adaptable Corpus Tools /Andriy Lutskiv, Nataliya Popovych// Computational Linguistics and Intelligent Systems. Proc. 4thInt. Conf. COLINS 2020. Volume I:Workshop. Lviv, Ukraine, April23-24, 2020, CEUR-WS.org, online. pp.374-395. [Electronic resource] Access mode: URL: http://ceur-ws.org/Vol-2604/

7. Lutskiv A. Big data-based approach to automated linguistic analysis effectiveness.//A. Lutskiv, N. Popovych/ IEEE Third International Conference on Data Stream Mining & Processing August 21-25, 2020, Lviv, Ukraine pp.438–443. DSMP 2020.

8. Lutskiv A. Corpus-Based Translation Automation in Adaptable Corpus Translation Module /Andriy Lutskiv, Roman Lutsyshyn// Computational Linguistics and Intelligent Systems. Proc. 5th Int. Conf. COLINS 2021. Volume I: Workshop. Lviv, Ukraine, April 22-23, 2021, CEUR-WS.org, online. pp.374-395. [Electronic resource] Access mode: URL: http://ceur-ws.org/Vol-2604/R. Malhotra, “A systematic review of machine learning techniques for software fault prediction,” Appl. Soft Comput.vol. 27. Feb. 2015. pp. 504–518

9. L. Son et al., “Empirical Study of Software Defect Prediction: A Systematic Mapping,” Symmetry (Basel)., vol. 11, no. 2. 2019. p. 212.

10. A. Hudaib et al., “ADTEM-Architecture Design Testability Evaluation Model to Assess Software Architecture Based on Testability Metrics,” J. Softw. Eng. Appl., vol. 08, no. 04. 2015. pp.201–210.

11. Яцишин В.В., Шуптарський В.В., Цісарук Д.А. Алгоритми машинного навчання для сегментації користувачів у маркетингових комп’ютерних систем. Матеріали X міжнародної науково - технічної конференції молодих учених і студентів «Актуальні задачі сучасних технологій» (24-25 листопада 2021 р.) Тернопільського національного технічного університету імені Івана Пулюя. Тернопіль: ТНТУ. 2021. С. 145.

12. Луцків А.М., Цісарук Д.А., Шуптарський В.В. Аналіз життєвого циклу процесу тестування програмного забезпечення комп’ютерних систем. Матеріали IX науково-технічної конференції Тернопільського національного технічного університету імені Івана Пулюя «Інформаційні моделі, системи та технології» (8-9 грудня 2021 року). Тернопіль: ТНТУ. 2021. С. 142.

13. E. Shihab et al., “Studying re-opened bugs in open source software,” Empir. Softw. Eng., vol. 18, no. 5. 2013. pp. 1005–1042.

14. M. Shepperd, Q. Song, Z. Sun, and C. Mair, “Data Quality: Some Comments on the NASA Software Defect Datasets,” IEEE Trans. Softw. Eng., vol. 39, no. 9, pp. 2013. 1208–1215.

15. Python-recsys on Github. URL: https://github.com/ocelma/python-recsys (дата звернення 22.10.2021 р).

16. Preprocessing data. URL: https://scikit-learn.org/stable/modules/ preprocessing.html#preprocessing (дата звернення 02.10.2021 р.).

17. API reference. URL: https://pandas.pydata.org/docs/reference/index.html (дата звернення 10.09.2021 р.).

18. NumPy Reference. URL: https://numpy.org/doc/stable/reference/index.html (дата звернення 12.10.2021 р.)

19. Shirkhorshidi A. S. Big data clustering: a review. International Conference on Computational Science and Its Applications. Springer, Cham, 2014. pp. 707-720.

20. Kurasova O. Strategies for big data clustering/ O. Kurasova et al. // 2014 IEEE 26th International Conference on Tools with Artificial Intelligence. IEEE, 2014. pp. 740-747.

21. Lilleberg, J.; Zhu, Y.; Zhang, Y. Support vector machines and Word2vec for text classification with semantic features. In Proceedings of the 2015 IEEE 14th International Conference on Cognitive Informatics & Cognitive Computing (ICCI*CC), Beijing, China, 6–8 July 2015. pp. 136–140.

22. 3. Lima, M.; Valle, V.; Costa, E.; Lira, F.; Gadelha, B. Software Engineering Repositories: Expanding the PROMISE Database; XXXIII Brazilian Symposium on Software Engineering; SBC: Porto Alegre, Brasil, 2020. pp. 427–436.

23. Zubcoff, J.J.; Garrigós, I.; Casteleyn, S.; Mazón, J.; Aguilar, J.A.; Gomariz-Castillo, F. Evaluating differenti*-based approaches for selecting functional requirements while balancing and optimizing non-functional requirements: A controlled experiment. Inf. Softw. Technol. 2019. 106, pp. 68–84.

24. Abad, Z.S.H.; Karras, O.; Ghazi, P.; Glinz, M.; Ruhe, G.; Schneider, K. What Works Better? A Study of Classifying Requirements. arXiv 2017, arXiv:1707.02358.

25. Микитишин А.Г., Митник М.М., Стухляк П.Д. Комплексна безпека інформаційних мережевих систем: навчальний посібник. Тернопіль: Вид-во ТНТУ імені Івана Пулюя, 2016. 256 с.

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

27. Катренко Л.А., Катренко А.В. Охорона праці в галузі комп’ютингу. Львів: Магнолія-2006. 2012. 544 с.

28. Желібо Е.Н. Безпека життєдіяльності: Навчальний посібник. Київ: «Каравела», Львів: «Новий світ - 2000». 2001. 320с.

Додаток А

Тези конференцій

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Тернопільський національний технічний університет імені Івана Пулюя (Україна)
Університет імені П'єра і Марії Кюрі (Франція)
Маріборський університет (Словенія)
Технічний університет у Кошице (Словаччина)
Вільнюський технічний університет ім. Гедимінаса (Литва)
Білоруський національний технічний університет (Республіка Білорусь)
Міжнародний університет цивільної авіації (Марокко)
Наукове товариство ім. Т.Шевченка

АКТУАЛЬНІ ЗАДАЧІ СУЧАСНИХ ТЕХНОЛОГІЙ

Збірник

тез доповідей

Том І

X Міжнародної науково-практичної конференції молодих учених та студентів

24-25 листопада 2021 року

УКРАЇНА
ТЕРНОПІЛЬ – 2021

Матеріали X Міжнародної науково-практичної конференції молодих учених та студентів
«АКТУАЛЬНІ ЗАДАЧІ СУЧАСНИХ ТЕХНОЛОГІЙ» – Тернопіль 24-25 листопада 2021 року

УДК 004.89

Яцишин В.В. канд. техн. наук, доцент, Шуптарський В.В., Цісарук Д.А.
Тернопільський національний технічний університет імені Івана Пулюя

АЛГОРИТМИ МАШИННОГО НАВЧАННЯ ДЛЯ СЕГМЕНТАЦІЇ КОРИСТУВАЧІВ У МАРКЕТИНГОВИХ КОМП'ЮТЕРНИХ СИСТЕМ

Yatsyshyn V.V. PhD, Assoc. Prof., Shuptarskyi V.V., Tsisaruk D.A.

MACHINE LEARNING ALGORITHMS FOR USER SEGMENTATION IN MARKETING COMPUTER SYSTEMS

Найбільш поширеними і широко використовуваними методами сегментації, які застосовуються при побудові маркетингових систем є групування за ознаками і методи статистичного аналізу.

В основі методу групування лежить принцип послідовного розбиття множини об'єктів на підмножини (групи) з врахуванням найбільш важливих властивостей чи ознак. Одна з таких властивостей об'єкту виділяється як критерій на основі якого можна сформувати деяку систему показників (тип продукції, виробник товару, потенційний споживач конкретного виду товару).

Побудова маркетингових комп'ютерних систем вимагає застосування сучасних методів і засобів, які повинні бути орієнтованими на вирішення задач аналізу поведінки користувачів, формування промоакцій, сегментації товарів, послуг і користувачів. Одними із таких трендових технологій для проведення маркетингових досліджень є методи машинного навчання, зокрема, кластеризація і класифікація користувачів і товарів.

У сфері машинного навчання кластеризація передбачає навчання без вчителя, тобто для цього типу алгоритму існує лише один набір вхідних даних без міток. Тому потрібно розв'язати задачу одержання інформації, не знаючи попередньо, яким буде вихід.

Кластеризація використовується в проектах для компаній, які хочуть виявити спільні властивості у своїх клієнтів, щоб застосувати сегментацію клієнтів, створити карти подорожей клієнтів або знайти групи та сформувати рекомендовані набори товарів чи послуг.

Таким чином, якщо значний відсоток клієнтів мають певні спільні риси (вік, тип сім'ї тощо), компанія може рекомендувати проведення певної кампанії, послуги чи товару. Кластеризація також корисна для одержання загальних уявлень про інформацію якою оперує компанія.

З іншого боку, класифікація належить до алгоритмів навчання з вчителем, тобто наявний контроль за навчанням. Це означає, що вхідні дані мають мітки класів і наперед відомий можливий вихід алгоритму. Розрізняють бінарну класифікацію, яка розв'язує задачі з категорійними відповідями (наприклад, "так" і "ні"), і мультикласифікація, для задач, де потрібно знайти більше двох класів, відповідаючи на більш відкриті відповіді, такі як «чудово », «звичайний» і «недостатній». Класифікація використовується в багатьох галузях, наприклад у біології або в десятковій класифікації Дьюї для книг, при виявленні спаму в електронних листах та ін. Класифікація зазвичай використовується у фінансовому секторі. В епоху онлайн-транзакцій, коли використання готівки помітно зменшилося, необхідно визначити, чи безпечні переміщення за допомогою карток. Суб'єкти можуть класифікувати операції як правильні або шахрайські, використовуючи історичні дані про поведінку клієнтів, щоб дуже точно виявляти шахраїв.

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

М А Т Е Р І А Л И

IX НАУКОВО-ТЕХНІЧНОЇ КОНФЕРЕНЦІЇ

«ІНФОРМАЦІЙНІ МОДЕЛІ, СИСТЕМИ ТА ТЕХНОЛОГІЇ»

8–9 грудня 2021 року

ТЕРНОПІЛЬ
2021

УДК 004.4

А.М. Луцків канд. техн. наук, доцент, Д.А. Цісарук, В.В. Шуптарський
(Тернопільський національний технічний університет імені Івана Пулюя, Україна)

АНАЛІЗ ЖИТТЄВОГО ЦИКЛУ ПРОЦЕСУ ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ КОМП'ЮТЕРНИХ СИСТЕМ

UDC 004.4

A.M. Lutskiv PhD, Assoc. Prof., D.A. Tsisaruk, V.V. Shuptarskyi

ANALYSIS OF SOFTWARE TESTING LIFE CYCLE PROCESS IN COMPUTER SYSTEMS

Життєвий цикл тестування програмного забезпечення комп'ютерних систем передбачає виконання різних комплексних заходів та видів діяльності, які формують деяку послідовність. Основна ціль підпроцесів життєвого циклу тестування полягає у забезпеченні та контролі якості прототипу системи та кінцевого програмного продукту. Структуру життєвого циклу процесу тестування програмного забезпечення показано на рис. 1.

Рисунок 1. Життєвий цикл тестування програмного забезпечення комп'ютерних систем

Фаза тестування при реалізації програмного забезпечення відіграє важливу роль у загальному життєвому циклі комп'ютерної системи. Команді тестувальників потрібно спланувати багато заходів і кожна діяльність має бути орієнтована на забезпечення якості.

Характерною і необхідною ознакою кваліфікованої команди тестувальників є володіння інформацією щодо масштабу і повноти виконання подальших кроків щодо виявлення дефектів. Аналогічно, на ранній стадії тестування повинна бути можливість аналізу обсягу тестування продукту. Кожен документ щодо підготовки і проведення різних видів діяльності у процесі тестування повинен бути підготовлений належним чином і однаково трактуватись усіма учасниками проекту.

Якщо всі тестові сценарії охоплені належним чином у відповідності до вимог, то виконання тестового сценарію займатиме менше часу. Це допоможе виявляти помилки на ранній стадії.