Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет _____Комп'ютерних наук_____
(повна назва)
Центр післядипломної освіти

Кафедра _____Програмної інженерії_____
(повна назва)

АТЕСТАЦІЙНА РОБОТА
Пояснювальна записка

_____другий (магістерський)_____
(рівень вищої освіти)

Дослідження методів та моделей комплектації команд розробників в професійних соціальних мережах
(тема)

Виконав: студент 2 курсу, групи ІПЗмзд-17-1___
спеціальності 121- Інженерія програмного забезпечення
(код і повна назва спеціальності)
Освітньо-наукової програми
Інженерія програмного забезпечення_____
(повна назва освітньої програми)

_____Костенко О. М._____
(прізвище, ініціали)
Керівник _____проф. Єрохін А.Л._____
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри, проф. _____ З.В.Дудар

2019 р.

Харківський національний університет радіоелектроніки

Факультет_ Центр післядипломної освіти_____

Кафедра_Програмної інженерії_____

Рівень вищої освіти_____другий (магістерський)_____

Спеціальність _121-Інженерія програмного забезпечення_____
(код і повна назва)
освітньо-наукова програма _Інженерія програмного забезпечення_____
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«___»_____20 ___ р.

ЗАВДАННЯ
НА АТЕСТАЦІЙНУ РОБОТУ

студентові _Костенко Оксані Миколаївні_____
(прізвище, ім'я, по батькові)

1. Тема роботи _Дослідження методів та моделей комплектації команд розробників в професійних соціальних мережах_____
затверджена наказом по університету від "22" квітня 2019 р № 65 Стз
2. Термін подання студентом роботи до екзаменаційної комісії 05 червня 2019 р.
3. Вихідні дані до роботи *методи вирішення задачі комплектації команд розробників, діаграма архітектури системи. Використовувати ОС Linux,_____ середовище об'єктно-орієнтованого проектування_____*
4. Перелік питань, що потрібно опрацювати в роботі *мета роботи, аналіз___ проблемної галузі і постановка задачі, огляд методів вирішення задачі_____ багатокритеріального вибору, аналіз можливих типів архітектури системи, побудування UML діаграм_____*

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) *Мета завдання, обґрунтування доцільності розроблення, постановка задачі, об'єктна модель системи, базові моделі, методи й алгоритми, UML-діаграми, архітектура системи,_____ демонстраційні матеріали*

6 Консультанти розділів роботи

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата
Спецчастина	проф. Єрохін А.Л.		

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка *
1.	Аналіз предметної галузі	19 квітня 2019р.	
2.	Огляд існуючих методів	27 квітня 2019р.	
3.	Методи розв'язання задачі	10 травня 2019р.	
4.	Підготовка пояснювальної записки	25 травня 2019р.	
5.	Спецчастина	26 травня 2019р.	
6.	Підготовка презентації та доповіді	28 травня 2019р.	
7.	Попередній захист	30 травня 2019р.	
8.	Нормоконтроль, рецензування	02 червня 2019р.	
9.	Занесення диплома в електронний архів	03 червня 2019р.	
10.	Допуск до захисту у зав. Кафедри	04 червня 2019р.	

Дата видачі завдання _11__ ____лютого___2019 р.

Студент _____
(підпис)

Керівник роботи _____ __проф. Єрохін А.Л._____
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ / ABSTRACT

Пояснювальна записка до атестаційної роботи: 54 с., 10 рис., 14 табл., 3 додатки, 27 джерел.

ПРОФЕСІЙНІ СОЦІАЛЬНІ МЕРЕЖІ, КОМПЛЕКТАЦІЯ РОЗРОБНИКІВ, ПРОГРАМНИЙ ПРОДУКТ, МЕТОДОЛОГІЯ.

Метою роботи є створення методології для системи комплектації команд розробників у професійних соціальних мережах.

Об’єктом дослідження є сучасні методи комплектації команд розробників. Основним завданням є дослідження системи комплектації команд розробників у професійних соціальних мережах.

У результаті роботи був проведений аналіз предметної галузі, виявлено проблемні місця та поставлено задачу розробки програмного забезпечення. Було сформовано вимоги до системи та досліджено методи комплектації команд розробників.

PROFESSIONAL SOCIAL NETWORKS, SOFTWARE, DEVELOPMENT MANUFACTURERS, METHODOLOGY.

The purpose of the work is to create a methodology for the system of complete set of teams of developers in professional social networks.

The object of the research is the modern methods of the complete set of teams of developers. The main task is to investigate a system of complete set of teams for developers in the profesional social networks.

As a result of work, the analysis of the subject area was carried out, problem areas were identified and the task of software development was set. The requirements for the system were formulated and methods for the development of the teams of teams were explored.

ЗМІСТ

Перелік умовних скорочень..... 6
Вступ..... 7
1 Аналіз предметної області та постановка задачі……..... 9
1.1 Опис предметної області. Поняття сучасних професійних соціальних мере……. 9
1.2 Аналіз методів формування команд розробників ……..……..... 12
1.3 Дослідження можливості комплектації команд розробників за допомогою професійних соціальних мереж……. 15
1.4 Постановка задачі. Актуальність роботи…….. 16
2 Опис моделі комплектації команд розробників..... 18
2.1 Загальний опис моделі що пропонується …..... 18
2.2 Вибір методу оцінки персональних якостей розробників……..... 19
2.3 Опис алгоритму комплектації команд розробників…..... 21
2.4 Опис основних функцій моделі…….……23
3 Розробка алгоритмічного та програмного забезпечення..... 28
3.1 Вербальний опис методу формування команд розробників…….. 28
3.2 Розробка специфікації системних вимог програмного забезпечення..... 29
3.3 Вибір системної архітектури та UML проектування……..... 32
3.4 Вибір інструментальних засобів реалізації програмного забезпечення…… 38
4 Програмна реалізація та експериментальне дослідження запропонованого підходу..... 44
4.1 Вимоги до програмного забезпечення…..... 44
4.2 Тестування системи……..... 45
4.3 Опис візуального інтерфейсу системи…….. 53
4.4 Апробація результатів роботи……56
Висновки..... 57
Перелік джерел посилання……..... 59
Додаток А Програмний код……..…..……..……..…….. 62
Додаток Б Слайди презентації.....……..…….. 64
Додаток В Апробація результатів роботи.....…….……..…… 71

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

БД – База Даних;

ООП – Об’єктно Орієнтоване програмування;

ОС – Операційна Система;

ПЕОМ – Персональна Електрона Обчислювальна Машина;

ПЗ – Програмне Забезпечення;

ПК – Персональній Комп’ютер;

ПС – Програмна Система;

API – Application Programming Interface;

CASE – Computer-Aided Software Engineering;

CLR – Common Language Runtime;

CPU – Central Processing Unit;

DDD – Design Diversity Dimensions;

GUI – Graphical User Interface;

HDD – Hard Drive Disk;

HTTP – HyperText Transfer Protocol;

IEEE – Institute of Electrical and Electronics Engineers;

KTS – Keirsey Temperament Sorter;

MBTI – Myers–Briggs Type Indicator

MMPI – Minnesota Multiphasic Personality Inventory;

MVC – Model-View-Controller;

NVP – N-Version Programming;

NVS – N-Version Software;

RAM – Random Access Memory;

SLOC – Source Lines of Code;

UML – Unified Modeling Language;

ВСТУП

Для роботи над складними і масштабними проектами необхідно формувати групи фахівців, що володіють необхідними якостями, навичками, знаннями і можливостями. Успішність колективної роботи визначається здатністю групи до колективного взаємодії, довірою і інтеграцією членів групи, а також їх професійними навичками [1]. Крім того, потрібен відповідний рівень згуртованості колективу, щоб команда не розпадалася, співпрацювала і таким чином, забезпечувалися умови для високої якості командної роботи [2].

Низький рівень взаємодії в групі може викликати зниження ефективності функціонування і вилитися в фінансові втрати підприємства. Саме тому однією з важливих проблем проектного менеджменту є необхідність ефективного підбору та управління роботою команди. Дана проблема вирішується як на найпершому етапі управління проектом, коли проектна команда знаходиться в стадії формування, так і в процесі роботи вже сформованої команди [3].

В даній роботі розглянуто питання формування команд розробників програмного забезпечення за допомогою професійних соціальних мереж. Така модель дозволяє формувати команду на основі даних, які знаходяться в соціальній мережі, таких як досвід, навички, відгуки тощо. Керуючись цими та подібними атрибутами система виконує розподіл розробників за командами. Дана робота надає метрики та модель для оцінки рівня суб’єктної диверсифікації спираючись на атрибути розробника, та на атрибути програмного коду що він виробляє. Висувається гіпотеза про зв’язок між профілем особистості розробника та набором певних атрибутів його професійних якостей.

Актуальність теми. Актуальність дослідження полягає в тому, що воно дозволяє формально вирішити задачу формування команд розробників програмного забезпечення та підвищити загальний рівень якості та швидкості створення програмного продукту за рахунок знаходження оптимальної конфігурації команд розробників.

Мета і задачі дослідження. Метою роботи є забезпечення можливості чисельного вимірювання підходящого кваліфікаційного рівня кандидата на основі персонального профілю розробника в професійній соціальній мережі.

Для безпосереднього виявлення набору кандидатів для розробки певного програмного проекту необхідно виконати наступний перелік задач дослідження: знайти або розробити засіб, який би дозволив виявляти певний набір якостей деяких кандидатів; отримати оцінки за певним набором якостей на певній множині кандидатів, згідно до висунутої гіпотези дослідження; отримати оцінки складності програмного продукту, що потребує розробки; оцінити кореляцію між рівнем складності програмного продукту та між оцінками за бажаним набором якостей кандидатів.

Об’єкт дослідження. Сучасні методи комплектації команд розробників.

Предмет дослідження. Проблема автоматизації комплектації команд розробників за допомогою професійних соціальних мереж.

Методи дослідження. Для досягнення мети застосовувались такі методи: Knowledge Diversity (для отримання індексу на основі показника ентропії); (Social category Diversity (для дослідження впливу впливу соціальної категорії); Value Diversity (для оцінки ставлення кандидата до того чи іншого питання); методи оцінки типу особистості та інші.

Наукова новизна одержаних результатів. Знайдено засіб, який би дозволив виявляти певний набір якостей деяких кандидатів (вперше одержано); отримано оцінки за певним набором якостей на певній множині кандидатів, згідно до висунутої гіпотези дослідження (вперше одержано); отримано оцінки складності програмного продукту, що потребує розробки (дістало подальшого розвитку.

Практичне значення одержаних результатів. Результати досліджень було запропоновано запровадити для однієї із популярних професійних соціальних мереж Skillsnet.

Публікації. Протягом всього часу дослідження результати були представлені в рамках 3-ї міжнародної науково-технічної конференції «Комп’ютерні та інформаційні системи і технології» [21].

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Опис предметної області. Поняття сучасних професійних соціальних мереж

Поява і розвиток соціальних мереж сприяє розвитку нової культури і усього суспільства. Такий спосіб комунікації виконує велику кількість функцій, дозволяє людині самореалізуватися, отримувати нову корисну інформацію. Але головною метою використання соціальних мереж є соціалізація і інтеграція, прагнення до встановлення стосунків з іншими користувачами. Будь-яка функція соціальної мережі переплітається з комунікаційною і, зрештою, трансформується в комунікацію.

Термін “соціальна мережа” ввів англійський соціолог Джеймс Барнс у 1954 р. Цим терміном він висловив думку про те, що суспільство – це складне переплетення особистісних соціокомунікаційних стосунків. Барнс досліджував взаємозв’язки між людьми (членами соціуму) за допомогою візуальних діаграм, в яких окремі особи зображались крапками, а зв’язки між ними – лініями [2].

Сучасне тлумачення цього поняття в означає певне коло знайомих людини, де центром цього кола, тобто соціальної мережі є сама людина, її знайомі – це гілки соціальної мережі, а відносини між цими людьми - зв'язки соціальної мережі. Якщо розглядати соціальну мережу більш глибоко, можна виявити, що зв'язки діляться за типами: односторонні і двосторонні; мережі друзів, колег, однокласників, однокурсників і т.д. У наш час соціальні мережі з звичайного професійного поняття соціологів перетворилося на сучасну концепцію, що є однією з центральних у концепцій веб 2.0.

Великі групи соціальних мереж - це комерційні, професійні та суспільно-орієнтовані групи, для порівняння та оцінки яких застосовні різні критерії та методи.

Всі соціальні мережі поділяються за типом. Існують мережі для пошуку людей: однокласників, однокурсників, колег та інших людей. Є бізнес-мережі, для пошуку роботи, партнерів, професійного спілкування та інших ділових питань.

В різних мережах по-різному ставляться до відкритості інформації. Більшість мереж на даний момент відкриті, але є і закриті, куди люди потрапляють тільки на запрошення. Закриті мережі тільки починають з'являтися, але вже можна чекати їх популярність через кілька років, людям, природно, подобається все заборонене і важкодоступне. Також умовно мережі можна розділити за географічною орієнтацією на світові, державні та регіональні.

Не зважаючи на різноманіття соціальних мереж, всі вони мають спільну властивість, що є суттєвою для бізнесу – це можливість зв’язків з широким колом клієнтів. Перевагою участі бізнесу у спілкуванні з клієнтами на просторах соціальних мереж є велика їх кількість у всесвітній мережі.

Сучасні технології змінили саме розуміння спілкування. Якщо раніше компанії відділялися і навіть в деякому сенсі нехтували соціальними мережами і взагалі Інтернетом, то зараз вони розробляють свої подібні Інтернет проекти. Перевагами професійних соціальних мереж є можливість за їх допомогою налагоджувати взаємовідносини з майбутніми співробітниками, а також здійснювати їх моніторинг.

Окрім мереж загальної тематики і спеціалізованих існують так звані ділові, що більш відповідають рівню організації, сприяючи професійному розвитку та побудові кар’єри особистості. Бурхливий розвиток професійних соціальних мереж викликаний в першу чергу можливостями, які вони надають для пошуку кандидатів. Основні переваги професійних соціальних мереж перед традиційними полягають в наступному [3, 7]:

– наявність інформації про потенційних кандидатів у професійних - соціальних мережах. Соціальна мережа вже наповнена інформацією про потенційних співробітників, їх досвід, навички тощо;

– наявність необхідних комунікаційних інструментів для роботи з кандидатами всередині соціальної мережі. У таких соціальних мережах не

потрібна спеціальна поштова програма для відправки повідомлень, оскільки інструменти листування користувачів мережі вбудовані в систему;

– гарантована доставка повідомлення до одержувача. Одержувач точно побачить повідомлення з мережі, воно не буде піддано і затримано спам-фільтрами;

– простота доступу до інформації. Якщо вам потрібно подивитися історію листування з конкретною людиною – достатньо просто перейти до його профілю;

– швидкість поширення інформації. Плануєте провести набір співробітників напишіть;

– низька вартість. Тут соціальним мережам точно немає конкурентів. Реєстрація та більшість можливостей безкоштовні.

Найпоширенішими професійними соціальними мережами вважаються:

– Linkedin.com – соціальна мережа для пошуку ділових контактів;

– Мой Круг – пошук друзів, роботи і співробітників;

– ВШтаті – універсальний сайт пошуку людей, резюме, компаній, вакансій;

– XING.com – професійна мережа з пошуку роботи і встановлення контактів;

– Skillsnet – професійна соціальна мережа для пошуку ділових контактів.

Такі мережі є популярними серед спеціалістів з підбору персоналу (HR). Вони у певній мірі є аналогом резюме, що дозволяє користувачу знайти роботу, а роботодавцю - працівника з відповідними функціональними можливостями.

1.2 Аналіз методів комплектації команд розробників

Для роботи над складними і масштабними проектами необхідно формувати групи фахівців, що володіють необхідними якостями, навичками, знаннями і можливостями. Успішність колективної роботи у чому визначається здатністю групи до колективного взаємодії, довірою і інтеграцією членів групи, а також їх професійними навичками [1]. Крім того, потрібен відповідний рівень згуртованості колективу, щоб команда не розпадалася, співпрацювала і таким чином, забезпечувалися умови для високої якості командної роботи [2].

Низький рівень взаємодії в групі може викликати зниження ефективності функціонування і вилитися в фінансові втрати підприємства. Саме тому однією з важливих проблем проектного менеджменту є необхідність ефективного підбору та управління роботою команди. Дана проблема вирішується як на найпершому етапі управління проектом, коли проектна команда знаходиться в стадії формування, так і в процесі роботи вже сформованої команди [3].

Зарубіжна соціальна психологія розглядає кілька підходів до командоутворення:

– розвиток і погодження цілей команди, де командоутворення визначають як становлення здатності груп людей досягати своїх цілей;

– інтерперсональний підхід, який сконцентрований на аналізі процесів і поліпшення міжособистісних відносин (зростання зв'язків в групі, за рахунок чого команда здатна працювати з більшою ефективністю);

– рольовий підхід, відповідно до якого командоутворення є вдосконаленням роботи команди через підвищення ясності ролей, в результаті чого всі її члени знають свою роль і роль інших, очікування по відношенню до інших і інших щодо нього;

– рішення проблем, коли команда має більш високу ефективність в ході спільного вирішення проблем усіма членами (вони визначають основні проблеми, вирішують їх і активно планують роботу).

Технологія формування команди включає в себе послідовне проходження трьох етапів, на кожному з яких необхідно відповісти на ряд питань (див. табл. 1.1).

Таблиця 1.1 – Стадії формування команд

Стадії формування команд	Проблеми стадії
Задачі	Яка робота повинна бути виконана? Які повноваження будуть у групи для управління своєю роботою? Що являє собою центральний аспект роботи, що виконується групою? Який ступінь залежності учасників команди один від одного? Є тільки єдине правильне рішення або можна знайти кілька можливостей? Чи збігаються інтереси членів команди або мають конкурентний характер?
Люди	Скільки людей буде в команді? Хто ідеально підійде для виконання роботи? Який рівень технічних навичок, навиків управління завданнями і міжособистісних навичок потрібно? Який тип і рівень диференційованості можна вважати для команди оптимальним?
Взаємовідносини	Яким чином учасники команди пристосовуються один до одного? Як виглядає неявне розподіл ролей між учасниками? Які норми можна вважати сприятливими, а які небезпечними для групи? Чи відіграє роль згуртованість команди? Як зростає довіра, що може йому загрожувати і як його встановити між учасниками?

При комплектуванні команди важливо знання принципів і умов її формування. Як вважав Р. М. Белбін, частіше команда формується з менеджерів, які виконують відповідні обов'язки в своїх підрозділах. Вони стають учасниками команди завдяки виконуваних функцій. Тут головну роль відіграє психологічна сумісність учасників команди.

Згуртована команда формується не в будь-якому випадку і не з будь-яких співробітників. Необхідно взяти до уваги кілька умов:

– учасники повинні бути фахівцями, виступаючи експертами при вирішенні покладених на них завдань;

– спільний досвід і таланти учасників команди повинні перевищувати досвід і здібності кожного з тих, хто працює в поодинці;

– велика частина учасників повинна мати можливість впливу на прийняття рішень, які їм потрібно виконувати, що підвищує інтерес до спільної справи;

– кожен учасник повинен мати схильністю до творчості для систематичного використання даного потенціалу в роботі групи.

Максимально докладно етапи і процеси командоутворення описала нормативна модель командоутворення, розроблена Жуковим, Журавльовим і Павлової. Вони розглядали кілька етапів створення команди.

1. Комплектування (перекомплектування), що означає визначення чисельності і складу учасників (майбутніх членів) команди.

2. Знайомство і його поглиблення з установкою первинного контакту, необхідного рівня довіри між учасниками. Тут члени команди знайомляться і відбувається орієнтуванні один в одному і в ситуації.

3. Інституалізація, відповідно до якої команда вписується в структуру і систему організації. Вона повинна володіти правами та обов'язками, доступом до інформаційних та інших ресурсів. Важливо встановити і, якщо це потрібно, документально оформити порядок роботи, методи взаємодії, визначити коло осіб, які курирують групу і координують роботу з іншими командами, а також структурними підрозділами та організаційними службами.

4. Встановлення загального бачення, що має на меті узгодити погляди, позиції, образу бажаного майбутнього у учасників командоутворення.

5. Позиціонування (перепозиціювання), яке представляє собою установку місця людини в ділових і персональних відносинах (підпорядкованість, звітність, відповідальність, дистанція) в контексті організації.

6. Планування першого кроку зі створенням целереалізующіх системи команди і складанням конкретного плану-графіка, розподілом ресурсів і відповідальності.

7. Виконання, де здійснюють те, що намітили і спланували раніше.

8. Рефлексія з безперервним відстеженням командою ступеня ефективності в русі вперед. Тут учасниками проводиться оцінка виконання всіх завдань, аналіз перешкод і можливостей, завдяки яким команда може працювати продуктивно.

9. Планування другого кроку, враховуючи аналіз нового положення справ.

Розглянута нормативна модель працює як засіб орієнтації в діях, які сконцентровані на надання роботі групи елементів командності. Модель розглядає етапи командоутворення по відношенню до того, які процеси домінують на кожному етапі.

1.3 Дослідження можливості комплектації команд розробників за допомогою професійних соціальних мереж

В сучасному глобалізованому світі питання, пов’язані із створенням команди, надзвичайно актуальні. Одна, навіть геніальна людина, просто група людей не спроможні швидко відреагувати на зміни у бізнесі, політиці, економіці. Саме тому, в наш час створення команди є найбільш прогресивною стратегією організації [6]. Сильна, єдина команда – основа ефективної діяльності підструктур організації та організації в цілому.

Команда – це колектив фахівців, які поділяють цілі, цінності та загальні підходи до реалізації спільної діяльності, мають взаємодоповнювальні навички, беруть на себе відповідальність за кінцеві результати. Стабільність роботи організації, її ефективність діяльності – це не сума діяльності кожного із співробітників окремо, це робота команди.

Командоутворення (англ. ”teambuilding”) – побудова ефективної команди, створення групи людей націлених на досягнення єдиного результату, працюючої злагоджено, як єдиний організм; процес цілеспрямованого формування особливого способу взаємодії людей в організації, що дозволяє ефективно реалізовувати їх енергетичний, інтелектуальний і творчий потенціали [40].

Суть командоутворення в організації полягає у створенні необхідних умов для формування навичок ефективної роботи в команді, формування команди та

вироблення командного духу, вміння працювати в команді, виявлення лідерів згуртування колективу і створення атмосфери неформального спілкування.

Значну допомогу у формуванні команди можуть надати професійні соціальні мережі. Вони можуть бути використані для спрощення деяких з етапів процесу найму нового співробітника. Сьогодні відділи HR по всьому світу використовують соціальні мережі як один з основних інструментів рекрутменту, а мільйони працівників переглядають спеціалізовані та корпоративні сайти в пошуках цікавих для них вакансій.

Але, при великій кількості професійних соціальних мереж, все ж таки пошуки кандидатів займають дуже багато часу, тому виникає необхідність автоматизувати цей процес шляхом запровадження технології с професійну соціальну мережу, яка б дозволила зібрати команду кандидатів, виходячи з їх професійного досвіду та навичок.

Аналізуючи існуючі професійні соціальні мережі, не було виявлено жодної, яка би дозволила автоматизувати цей процес, хоча є наявність усіх необхідних даних про кандидатів.

1.4 Постановка задачі. Актуальність роботи

Об’єктом дослідження є процес розробки відмовостійкого програмного забезпечення.

Предмет дослідження – проблема автоматизації комплектації команд розробників за допомогою професійних соціальних мереж.

Гіпотеза дослідження полягає в припущенні, що рівень різноманітності команди розробників залежить від особистих якостей розробників (профілю), що визначаються одним з доступних методів.

Ціллю дослідження є забезпечення можливості чисельного вимірювання підходящого кваліфікаційного рівня кандидата тільки на основі персонального профілю розробника.

Для безпосереднього виявлення набору кандидатів для розробки певного програмного проекту необхідно виконати наступний перелік задач дослідження:

– знайти або розробити засіб, який би дозволив виявляти певний набір якостей деяких кандидатів;

– отримати оцінки за певним набором якостей на певній множині кандидатів, згідно до висунутої гіпотези дослідження;

– отримати оцінки складності програмного продукту, що потребує розробки;

– оцінити кореляцію між рівнем складності програмного продукту та між оцінками за бажаним набором якостей кандидатів.

Актуальність дослідження полягає в тому, що воно дозволяє формально вирішити задачу формування команд розробників програмного забезпечення та підвищити загальний рівень якості та швидкості створення програмного продукту за рахунок знаходження оптимальної конфігурації команд розробників.

2 ОПИС МОДЕЛІ КОМПЛЕКТАЦІЇ КОМАНД РОЗРОБНИКІВ

2.1 Загальний опис моделі що пропонується

Гіпотеза дослідження полягає в припущенні, що існує зв’язок між набором певних особистих якостей розробника (профілем) та способом взаємодії у команді при розробці програмного забезпечення. Таким чином, для перевірки гіпотези дослідження необхідно:

– вибрати метрики оцінки профілю розробника;

– вибрати метрики оцінки способу розробки програмного забезпечення розробником (показники диверсифікації);

– зібрати статистичні дані: профілі розробників та зразки програмного коду продукту;

– оцінити наявність кореляції між узагальненими оцінками типу розробника та способом розробки.

Якщо гіпотеза дослідження вірна, можна запропонувати такий метод формування команд розробників:

– формування команд розробників:

– набір виконавців;

– оцінка профілю розробника;

– сортування розробників за профілем: найбільш схожі профілі мають бути поміщені в одну команду.

– розробка програмного забезпечення за звичайною моделлю життєвого циклу;

– оцінка отриманого рівня якості ПЗ та порівняння з раніше прогнозованим.

Дану роботу спрямовано саме на перевірку гіпотези дослідження та представлення попередніх результатів, що потребує розробки спеціалізованого програмного забезпечення, отримання даних від респондентів, тощо.

2.2 Вибір методу оцінки персональних якостей розробників

Оцінка персональних якостей може відбуватися за допомогою визнаних методів діагностики типу особистості. Деякі з них представлені в таблиці 2.1.

Таблиця 2.1 – Деякі методи оцінки типу особистості

Назва	Кількість шкал	Індекс процедури
Minnesota Multiphasic Personality Inventory (MMPI)	10 + 4	MeSH F04.711.647.513.607
16 Personality Factor Questionnaire (16PF)	16	MeSH F04.711.647.513.146
Myers–Briggs Type Indicator (MBTI)	4	-
Keirsey Temperament Sorter (KTS)	4	-
Eysenck Personality Questionnaire	3	-
Lüscher color test	1 + 1 + 5	-

Як видно з таблиці 2.1, деякі методи відносяться до медичних процедур, тому використовувати належним чином їх можна тільки із залученням кваліфікованого 16 лікаря. Інші методи, такі як Lüscher color test, вимагають покупки офіційного стимульного матеріалу [10]. 2.3.1 Метод Myers–Briggs Type Indicator (MBTI) Типологія Майєрс-Бріггс – типологія особистості, що виникла на базі типології Юнга в 1940х роках і набула широкого поширення в США і Європі. На основі цієї типології була створена система психологічного тестування – MyersBriggs Type Indicator (MBTI). Академічна психологія ставиться до цього підходу насторожено, приймаючи його з численними застереженнями, так як не всі вихідні припущення теорії були потім підтверджені після накопичення емпіричних даних. Ідентифікатор типів Майєрс-Бріггс широко застосовується в

бізнесі, зокрема, в ряді великих західних компаній. У США до 70 відсотків випускників середніх шкіл проходять визначення типу особистості за допомогою MBTI для цілей вибору майбутньої професії. Щорічно понад 2 млн осіб заповнюють опитувальник MBTI. Опитувальник MBTI переведений на 30 мов (в тому числі і українську мову) і використовується по всьому світу [11]. Основні прикладні області застосування типології Майерс-Бріггс [12]: – самопізнання і особистісний ріст; – кар'єрний ріст і профорієнтація; – розвиток організацій; – тренінги управління і лідерства; – вирішення проблем; – сімейні консультації; – освіта і складання навчального плану; – наукова робота; – тренінги міжособистісної взаємодії. 2.3.2 Метод Eysenck Personality Questionnaire (EPQ) Метод був запропонований в 1991 році, він призначений для вивчення індивідуально-психологічних рис особистості з метою діагностики ступеня вираженості властивостей, висунутих в якості істотних компонентів особистості: нейротизм, екстраверсії, інтроверсії і психотизма. Тест EPQ складається з 101 питання, які згруповані по 4 шкалами: екстраверсія - інтроверсія, нейротизм, психотизм і так звана шкала щирості [11]. 17 2.3.3 Метод Keirsey Temperament Sorter (KTS) Метод Keirsey Temperament Sorter (KTS) розроблено для самооцінки особистості та являється покращеною версією MBTI [13]. Це одна з найбільш широко використовуваних методик оцінки особистості в світі, його використовує Банк Америки, Allstate, ВПС США, IBM, AT&T, Coca-Cola. В той час як методика MBTI в основному зосереджена на тому, як люди думають і відчувають, метод KTS сфокусований на поведінці, яку можна спостерігати. Перелік показників, за якими KTS надає інформацію про респондента, показано в таблиці 2.2.

Таблиця 2.2 – Показники, за якими KTS надає інформацію про респондента

Назва метрики	Можливі значення
Extraversion	[0; 10]
Introversion	[0; 10]
Sensing	[0; 20]

Закінчення таблиці 2.2

Intuition	[0; 20]
Thinking	[0; 20]
Feeling	[0; 20]
Judging	[0; 20]
Perceiving	[0; 20]

Для спрощення дослідження в даній роботі автор пропонує використовувати широко доступний Keirsey Temperament Sorter, заснований на методиці Myers– Briggs Type Indicator. Цей метод є більш сучасним, порівняно з MBTI, він надає більше інформації, порівняно з EPQ, а його стимульний матеріал переведений на багато мов світу та знаходиться у вільному доступі.

2.3 Опис алгоритму комплектації команд розробників

Завдання формування команди проекту розглянемо в наступній постановці. Відомо безліч кандидатів в команду проекту (2.1).

$$X = \{x_j\}, j = \overline{1,m}, \tag{2.1}$$

Кожен кандидат x_j з безлічі X описується за допомогою узагальненої оцінки $E(x_j)$, отриманої на основі аналізу його професійних і особистісних якостей [4]. Необхідно визначити склад учасників команди проекту, які забезпечать максимальну ефективність вирішення завдань проекту.

Цільова функція даного завдання може бути представлена наступним чином (2.2).

$$F(k,t,t_{a\partial}) \to \max,$$
$$\begin{cases} f(k) \to \max, \\ t \to \min, \\ t_{a\partial} \to \min, \end{cases} , \qquad (2.2)$$

де

$f(k)$ - це функція якості виконаного проекту;

t - час, витрачений на виконання проекту;

$t_{a\partial}$ - час адаптації учасників команди, що виконує проект.

При оцінці кожного кандидата враховуються його комунікативні здібності, рівень інтелекту і параметри анкетних даних. Чим вище оцінка кандидата, тим вище ймовірність його відбору в сформовану команду (2.3).

$$E(x_i) = F(c(x_i), il(x_i), pd(x_i)) \to \max_{x_i \in x} \qquad (2.3)$$

де

$E(x_i)$ - узагальнена оцінка кандидата;

x_i; $c(x_i)$ - оцінка рівня комунікативних здібностей кандидата;

x_i; $il(x_i)$ - оцінка рівня інтелекту кандидата;

x_i; $pd(x_i)$ - сумарна оцінка професійних якостей кандидата, отримана на основі його анкетних даних x_i.

Рівень комунікативних здібностей кандидата повинен прагнути до максимуму. Чим вище рівень комунікативних і організаторських здібностей кандидата, тим менше часу знадобиться йому для адаптації в команді. якщо $c(x_i) \to \max$, то $t_{a\partial} \to \min$, що є першою умовою загальної постановки завдання.

Рівень інтелекту кандидата повинен прагнути до максимуму. Чим вище рівень інтелекту кандидата, тим ефективніше і швидше він буде виконувати свої

обов'язки в команді при роботі над проектом. Тобто, якщо $il(x_i) \to \max$, то $t \to \min$, що є другою умовою загальної постановки завдання даної роботи.

При визначенні рівня сумісності кандидатів використовуються узагальнені оцінки кандидатів. Чим вище узагальнені оцінки кандидатів, тим більше коефіцієнт їх сумісності і тим сильніша команда буде сформована при їх сукупності (2.4).

$$cc = \sum_{i=1}^{n} E(x_i) \to \max_{x_i \in x}, \tag{2.4}$$

де

CC - оцінка сумісності учасників команди;

n – загальна кількість учасників команди проекту.

2.4 Оцінка ефективності методу формування команд розробників

Пропонуємо оцінювати ефективність формування команд розробників за допомогою професійних соціальних мереж емпіричнім шляхом – через вимірювання рівня диверсифікації отриманого програмного забезпечення.

Для оцінки проекту необхідно використовувати тестування з урахуванням всіх версій даного програмного продукту. Аналіз отриманих результатів тестування з використанням спеціальних методів, надає можливість вимірювання рівня диверсифікації програмного забезпечення, що на практиці означає рівень незалежності програмного забезпечення від збоїв.

В ході аналізу поданих нижче методів з’ясувалося, що результати, які вони надають, математично не залежать від складу та способу формування команд розробників, тож можуть використовуватися для оцінки ефективності методу

формування команд за допомогою автоматизованого підбору в професійних соціальних мережах.

Метод Літлвуда та Міллера був представлений в 1989 році [15], він є одним з фундаментальних методів розробки тому що математично доводить потенційну можливість розробки програмного забезпечення з негативною кореляцією між кількістю версій та кількістю збоїв.

Цей метод надає такі важливі показники як:

– E(θ) – ймовірність збою випадково вибраної версії ПЗ на випадково вибраних вхідних даних;

– E(θ2) – ймовірність одночасного збою двох випадково вибраних версій ПЗ на випадково вибраних вхідних даних.

Для обчислення даних показників необхідно:

– скласти M наборів вхідних даних;

– провести тестування всіх N версій програмного забезпечення на всіх наборах вхідних даних;

– звести результати тестування в таблицю, приклад якої показано в таблиці 2.4, використовуючи функцію ν, яку наведено в формулі (2.5);

Зведення результатів тестування в таблицю виконується за допомогою формули (2.5).

$$v(\pi, x) = \begin{cases} 1, \text{якщо версія } \pi \text{ дає збій на данних } x \\ 0, \text{інакше.} \end{cases} \quad (2.5)$$

Приклад таблиці що містить результати тестуваня 5 версій ПЗ на 10 наборах вхідних даних наведені в таблиці 2.3.

Таблиця 2.3 – Приклад таблиці, складеної за результатми тестування 5 версій ПЗ

Вхідні дані, №	Версія, №					Всього за рядком
	1	2	3	4	5	
1	1	1	1	1	1	5
2	1	0	1	1	0	3

Закінчення таблиці 2.3

3	0	0	0	0	0	0
4	1	1	1	1	1	5
5	1	0	0	0	0	1
6	1	1	1	1	1	5

На основі таблиці 2.3 можна обчислити необхідні показники диверсифікації, як показано в формулі (2.6).

$$E(\theta) = \frac{1}{N}\frac{1}{M}\sum_N \sum_M v(\pi, x),$$

$$E(\theta^2) = \frac{1}{M}\sum_M \left(\sum_N v(\pi, x)\frac{1}{N}\right)^2, \quad (2.6)$$

де

N – кількість версій системи;

M – кількість наборів вхідних данних.

Якщо застосувати формули (2.6) до результатів тестування, що подані в таблиці 2.3, можна отримати такі результати: $E(\theta) = 0.52$, $E(\theta^2) = 0.44$. За цими значеннями можна зробити висновок, що п’ять версій програмного забезпечення з наведеного прикладу не забезпечують незалежності від збоїв. Ступінь відхилення від такої незалежності обчислюється за допомогою метрики $Var(0)$ за формулою.

$Var(0)$ – ступінь відхилення від незалежної поведінки [15], показник $Var(0)$ обчислюється за формулою (2.7).

$$Var(0) = E(\theta^2) - E(\theta)^2 . \quad (2.7)$$

Для показаного прикладу $Var(0) = 0.44 - 0.27 = 0.17$. З опису формул очевидно, що чим менше значення показників $E(\theta), E(\theta^2)$ – тим більше рівень незалежності від збоїв та, відповідно, менше значення $Var(0)$.

Метод Distinct Failure Diversity.

Вище було наведено приклад методу, що надає метрику незалежності N-версійного програмного забезпечення від збоїв, $Var(0)$. Однак, наведений раніше метод, має суттєвий недолік: всі збої програмного забезпечення, що тестується, розглядаються як ідентичні, проте насправді це можуть бути геть різні збої.

Для узагальненої оцінки досягнутої за рахунок багатоверсійності надійності, з урахуванням типів збоїв, можна використовувати критерій Distinct Failure Diversity (2.8) [16]:

$$DFD = \sum_{i=1}^{N} \frac{N-i}{N-1} k_i \in [0;1],$$
$$k_i = \frac{K_i}{K}. \qquad (2.8)$$

де

N – загальна кількість версій;

K_i – кількість ідентично провалених запусків i версіями;

K – загальне число унікальних провальних запусків.

В ході тестування вхідні та вихідні дані поділяють на категорії. Тоді, наприклад, для задачі розпізнавання букв українського алфавіту можна виділити 33 категорії вхідних даних та стільки ж категорій вихідних, при чому для кожної категорії j буде лише одна категорія правильних вихідних даних та 32 категорії неправильних.

Критерій *DFD* враховує саме унікальні помилки. Наведемо приклад розрахунку значення цього критерію для 4 категорій вхідних даних і 3 категорій вихідних. Відомості про результати тестування 5 версій деякого ПЗ подані в таблиці 2.4.

Таблиця 2.4 – Відомості про тестування 5-версійного програмного забезпечення на чотирьох категоріях вхідних даних, з трьома категоріями вихідних даних

Вхід	Вихід				Очікуваний вихід	Всього помилок
	1	2	3	4		
1	0	**2**	**2**	**1**	1	5
2	**2**	1	0	**2**	2	3
3	**1**	0	4	0	3	1
4	0	0	0	5	4	0

Напівжирним в таблиці виділено ненульові показники провальних запусків. Таким чином $K = 6$. K_1 буде дорівнювати числу провальних запусків, де показник кількості ідентично помилкових версій буде дорівнювати 1, тобто $K_1 = 2$. Аналогічно $K_2 = 4$, $K_3 = 0$, $K_4 = 0$, $K_5 = 0$.

Отримані значення k_i показані в таблиці 2.5.

Таблиця 2.5 – Обчислені значення k_i

i	k_i
1	$\frac{2}{6} = \frac{1}{3}$
2	$\frac{4}{6} = \frac{2}{3}$
3	0
4	0
5	0

Таким чином, показник *DFD* буде дорівнювати:

$$DFD = \frac{5-1}{5-1} \times \frac{1}{3} + \frac{5-2}{5-1} \times \frac{2}{3} = \frac{1}{3} + \frac{3}{4} \times \frac{2}{3} = \frac{10}{12} \approx 0.834.$$

Значення *DFD* змінюється в діапазоні $DFD \in [0; 1]$. Коли $DFD = 1$, всі провальні запуски були унікальними. Якщо кількість категорій вхідних даних $c \geq N$ і $DFD = 1$, можна стверджувати про відсутність диверсифікації.

3 РОЗРОБКА АЛГОРИТМІЧНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Вербальний опис методу формування команд розробників

Життєвий цикл розробки програмного забезпечення, як показано на рисунку, має ряд недоліків, таких як:

1) Склад команд розробників не задекларований як окреме вимірювання в множині DDD. Тим часом, одним з основних якостей N-версійного програмного забезпечення є спосіб його розробки, який передбачає роботу декількох ізольованих команд.

2) Формалізований спосіб формування диверсних команд розробників відсутній;

3) Розрізненість існуючих метрик оцінки суб’єктної диверсності не дозволяє однозначно вирішувати задачу формування команд.

Таким чином ми пропонуємо такі модифікації життєвого циклу N-версійного програмного забезпечення, що базується на вимірюванні суб’єктної диверсності розробників для формування диверсних команд:

1) диверсифікація команд розробників повинна розглядатися як вимірювання в множині DDD;

2) забезпечення достатнього рівня командної диверсифікації повинне базуватися на вимірюванні показників диверсифікації;

3) формування диверсних команд – окремий процес, що використовує результати вимірювання показників суб’єктної диверсифікації.

Як можна бачити, всі модифікації життєвого циклу N-версійного програмного забезпечення відносяться лише до початкових етапів розробки, що в подальшому спростить використання запропонованого життєвого циклу ПЗ.

3.2 Розробка специфікації системних вимог програмного забезпечення

Аналіз вимог полягає в визначенні потреб та умов які висуваються щодо нового, чи зміненого продукту, враховуючи можливо конфліктні вимоги різних замовників, таких як користувачі чи бенефіціари [1].

Аналіз вимог є критичним для успішної розробки проекту. Вимоги мають бути задокументованими, вимірними, тестовними, пов’язаними з бізнес-потребами, і описаними з рівнем деталізації достатнім для конструювання системи.

Функціональні вимоги – це перелік функцій або сервісів, які повинна надавати система, а також обмежень на дані і поводження системи при їхньому виконанні. Специфікація функціональних вимог – опис функцій та їхніх властивостей, які не містять у собі протиріч і виключень.

Для системи, що розробляється були сформульовані наступні функціональні вимоги:

1) засіб повинний мати змогу ідентифікувати користувачів;

2) засіб повинний мати змогу проводити тест MBTI над авторизованими користувачами;

3) засіб повинний мати змогу проводити юніт-тестування користувальницького коду в безпечному середовищі;

4) засіб повинний мати змогу обчислити значення метрик авторства програмного коду в безпечному середовищі;

5) засіб повинний мати змогу надавати результати тестування авторизованим користувачам;

6) засіб повинен працювати на більшості платформ без необхідності встановлення додаткового програмного забезпечення.

Функціональні вимоги зручно формалізувати у вигляді UML UseCase діаграм, які ще називають діаграмами прецедентів, це – граф, що складається з множини акторів, прецедентів (варіантів використання) обмежених границею

системи (прямокутник), асоціацій між акторами та прецедентами, відношень серед прецедентів, та відношень узагальнення між акторами [17]. Діаграми прецедентів відбивають елементи моделі варіантів використання. Суть даної діаграми полягає в наступному: проектована система представляється у вигляді безлічі сутностей чи акторів, що взаємодіють із системою за допомогою так званих варіантів використання. Варіант використання використовують для описання послуг, які система надає актору. Іншими словами, кожен варіант використання визначає деякий набір дій, який виконує система при діалозі з актором. При цьому нічого не говориться про те, яким чином буде реалізована взаємодія акторів із системою.

Функціональні вимоги було формалізовано до UseCase-діаграми, яку наведено на рисунку 3.1.

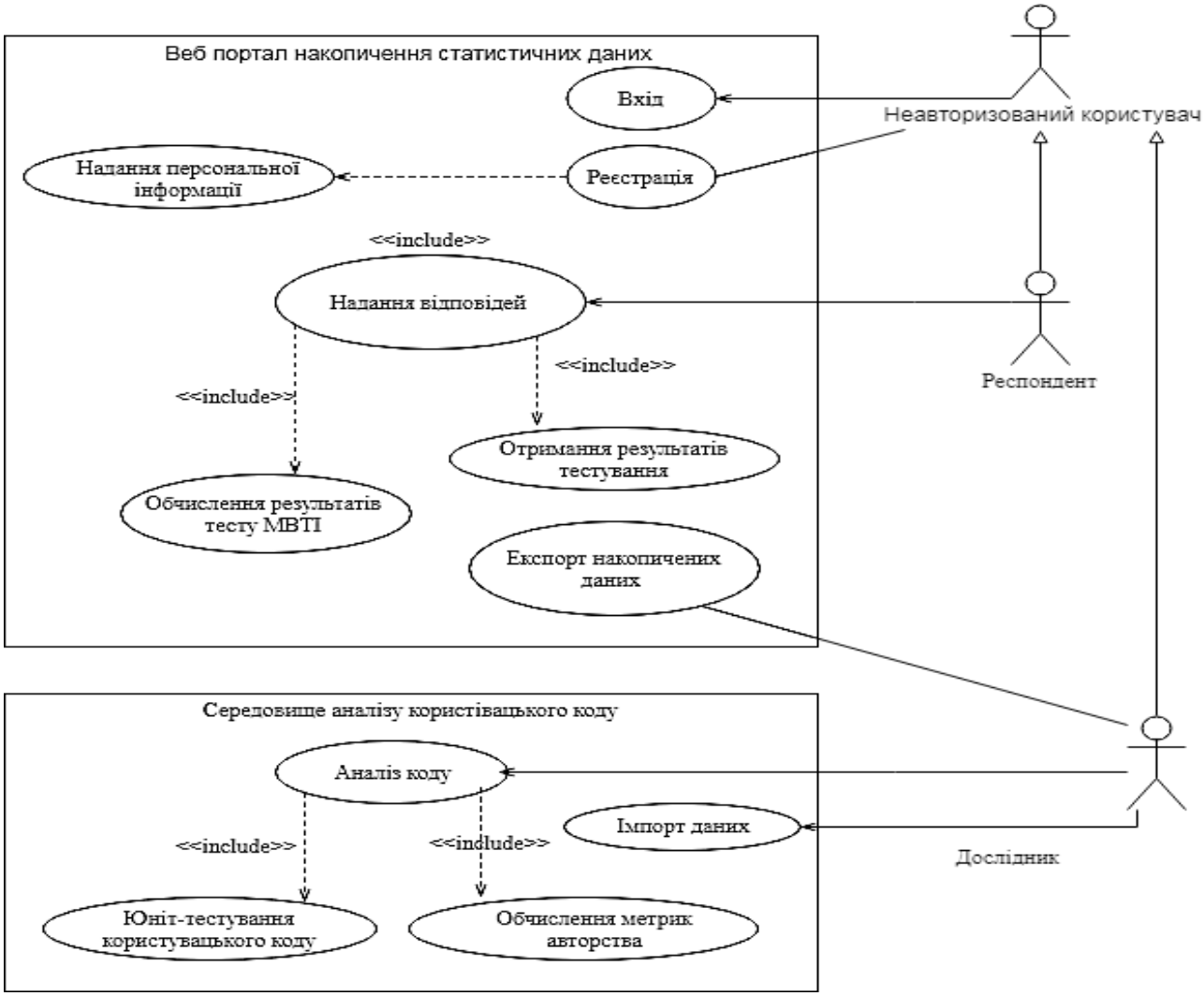

Рисунок 3.1 – UseCase-діаграма функціональних вимог

На діаграмі видно, що система повинна виконати юніт-тестування користувальницького програмного коду для визначення валідності цього коду. Це потенційно небезпечна процедура, адже юніт-тестування передбачає безпосереднє виконання користувальницького програмного коду. Таким чином, в цілях безпеки, юніт-тестування та аналіз користувальницького коду виконуються зовнішньою до публічно доступного сховища статистичних даних системою.

Нефункціональні вимоги визначають умови виконання функцій (наприклад, захист інформації у БД, аутентифікація доступу до ПС, тощо) у середовищі, що безпосередньо не пов’язані з функціями, а відбивають потреби користувачів щодо їх виконання. Ці вимоги характеризують принципи взаємодії із середовищами або іншими системами, а також визначають показники часу роботи, захисту даних і досягнення якості з урахуванням рекомендацій деякого стандарту [1]. Вони можуть встановлюватися як числові значення (наприклад, час чекання відповіді, кількість клієнтів, що обслуговуються і ін.) у різних одиницях виміру, включаючи, наприклад, ймовірність (значення ймовірності безвідмовної роботи системи – показника її надійності).

Для системи, що розробляється були сформульовані наступні не функціональні вимоги:

1) користувач повинен мати змогу використовувати ПЗ в зручному інтерфейсі;

2) програмне забезпечення повинне опрацьовувати дані без значних затримок в роботи графічного інтерфейсу користувача (до 7 секунд на будь-який запит).

3.3 Вибір системної архітектури та UML проектування

Архітектура програмного забезпечення містить такі важливі визначення [18]:

– організації системи програмного забезпечення;

– структурних елементів і їх інтерфейсів, з яких система складається, їх поведінка у взаємодії;

– склад цих елементів у великих підсистемах;

– архітектурний стиль, який визначає це ПЗ, його елементи і їх інтерфейси, їх взаємодія та їх склад.

Архітектура програмного забезпечення стосується не тільки структури та поведінки, а й використання, функціональності, продуктивності, стійкості, повторного використання, зрозумілості, економічних і технологічних обмежень,

компромісів і естетики ПЗ.

Системна архітектура – частина загальної архітектури ПЗ. Вона може бути представлена у вигляді кортежу, який задається наступною множиною:

$$SA = \langle C, F, I \rangle, \quad (8)$$

де

C – множина програмних компонентів, які реалізовані на різних мовах програмування та виконують задану функціональність системи;

F – множина допустимих структурних взаємодій, в яких можуть бути об’єднані елементи з множини C;

I – множина зовнішніх інтерфейсів, які дана система надає зовнішнім системам, компонентам, користувачам.

Автор розглядав наступні патерни системних архітектур:

– standalone application. Всі дані зберігаються та обробляються в одному місці;

– 2-tier application. Обробка даних може проходити завдяки серверу;

– 3-tier application. Обробка та збереження даних забезпечується відповідними серверами.

Було обрано трирівневу архітектуру завдяки наступним перевагам:

1) вимоги до програмного забезпечення декларують можливість роботи на широкому спектрі платформ, яку при невеликих вкладеннях ресурсів може забезпечити тільки трирівнева архітектура;

2) необхідно забезпечити максимальну простоту роботи з додатком без встановлення окремого програмного пакета, що також досягається завдяки використанню веб-технологій та трирівневої архітектури.

Для моделювання процесу виконання операцій у мові UML використовують діаграму діяльності, яка зображається графом, вершинами якого є стани (дій або видів діяльності), а дугами – переходи від одного стану до іншого стану [17].

Головним напрямом використання діаграм діяльності є візуалізація особливостей реалізації операцій класів, коли необхідно зобразити алгоритми їхнього виконання. При цьому кожен стан дії може бути виконанням операції деякого класу, а вид діяльності – послідовністю таких дій.

На рисунку 3.2 показано узагальнену діаграму діяльності методу що пропонується.

Діаграма діяльності на рисунку 3.2 репрезентує модель життєвого циклу для N-версійного програмного забезпечення. Її розроблено на основі моделі життєвого циклу поданого на рисунку А.1. Порівняно із нею, подана діаграма має суттєві відміни лише на початкових етапах розробки N-версійного програмного забезпечення:

– диверсні команди розробників розглядаються як ресурс – компонент множини DDD;

– вимірювання показників диверсифікації задекларовано як обов’язковий процес перед етапом верифікації та прийомки ПЗ для забезпечення зворотнього зв’язку з процесом формування диверсних команд, адже на вказаному етапі може бути обчислений очікуваний рівень диверсифікації;

– запропоновано процес уточнення кореляції між якостями (профілем) розробників та досягнутим рівнем диверсифікації для подальшого використання в майбутніх проектах (або ітераціях розробки).

Рисунок 3.2 – Узагальнена діаграма діяльності для методу формування команд розробників

Діаграма розгортання – діаграма, на якій представлені вузли системи [17].

Діаграма розгортання застосовується для подання загальної структури і топології системи і містить зображення розміщення компонентів по окремих вузлах системи. Крім того, діаграма розгортання показує наявність фізичних з’єднань – маршрутів передачі інформації між апаратними засобами, задіяними в реалізації системи.

На рисунку 3.3 показана діаграма розгортання для системи, що розроблюється.

Як видно з рисунка 3.3, для реалізації поданих вище вимог було обрано класичну трирівневу архітектуру з ”тонким” клієнтом – прикладним додатком, який базується на користувальницькому обладнанні.

На рисунку 3.3 є такі позначення що потребують пояснень:

– пристрій «клієнт» – будь-який пристрій що має змогу отримани дані за допомогою браузера по протоколу HTTP(s);

– «Застосування Web-портал» – веб-інтерфейс серверу накопичення статистичних даних та API для серверу аналізу користувальницького коду;

– компонент «Vagrant» – система управління віртуалізацією стороннього виробника що дозволяє запустити одразу декілька ідентичних віртуальних середовищ для завантаження, тестування, аналізу користувальницького коду.

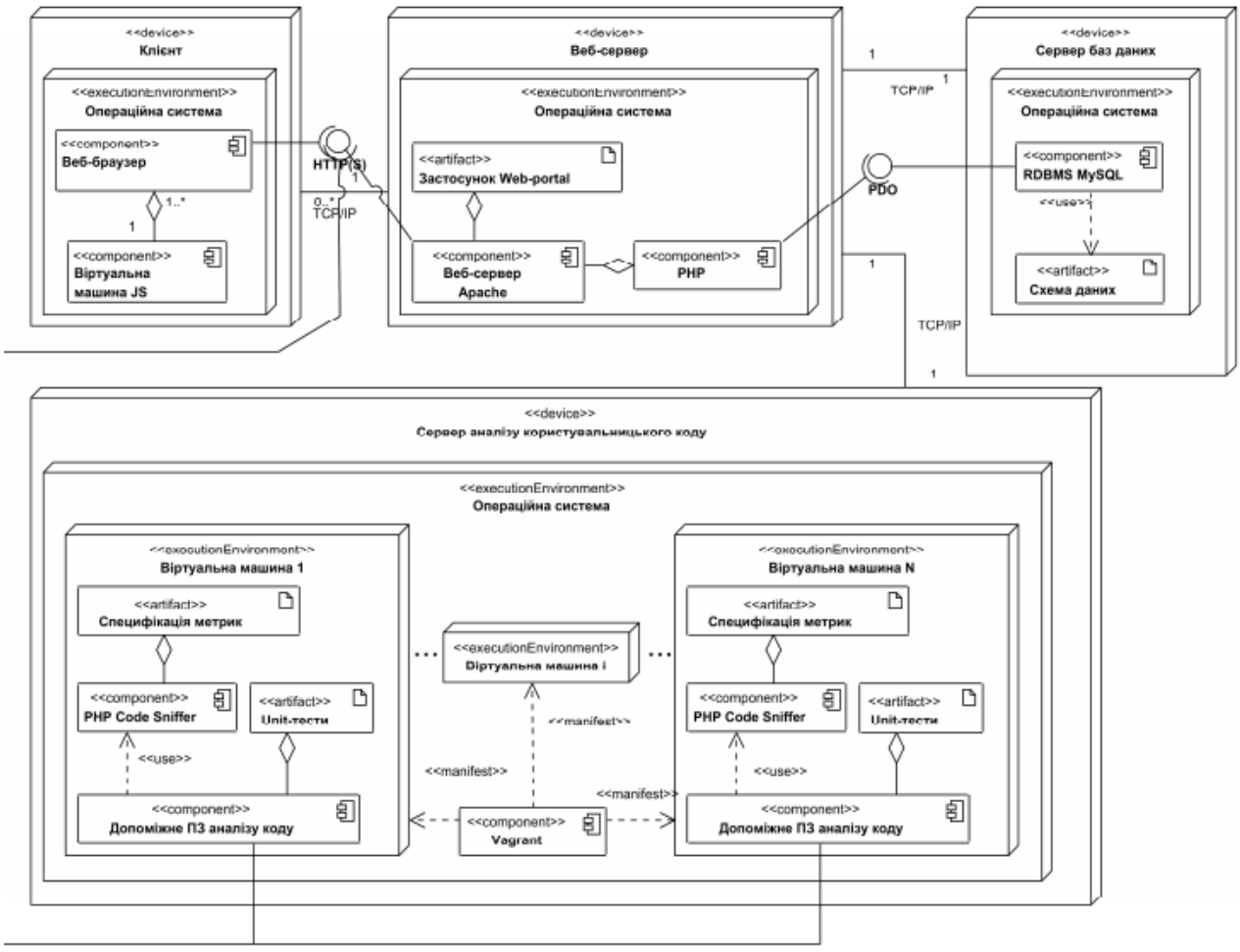

Рисунок 3.3 – Діаграма розгортання програмного забезпечення роботи

Діаграма класів — статичне представлення структури моделі. Відображає статичні (декларативні) елементи, такі як: класи, типи даних, їх зміст та відношення [17]. Діаграма класів, також, може містити позначення для пакетів та

може містити позначення для вкладених пакетів. Також, діаграма класів може містити позначення деяких елементів поведінки, однак їх динаміка розкривається в інших типах діаграм.

Лише частина програмного забезпечення роботи є об’єктно-орієнтованою та потребує декларування на діаграмі класів. На рисунку 3.3 об’єктно-орієнтованим програмним забезпеченням є «Застосування Web-portal», «PHP Code Sniffer» (бібліотека стороннього виробника), «Специфікація метрик» (модуль до бібліотеки PHPCS). Таким чином, декларування на діаграмі класів потребує лише артефакт «Застосування Web-portal». Його узагальнену діаграму класів показано на рисунку 3.4.

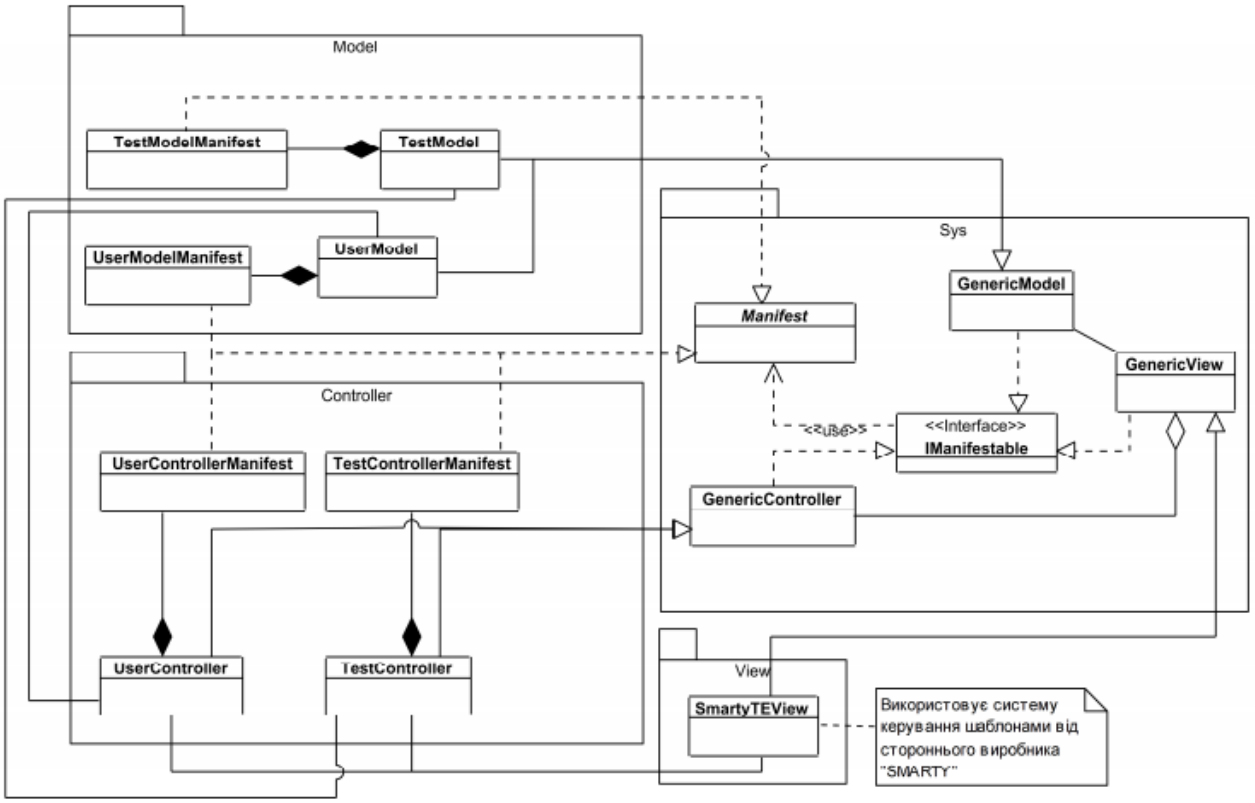

Рисунок 3.4 – Узагальнена діаграма класів застосування web-порталу

Як можна бачити з рисунку 3.4, програмне забезпечення реалізує патерн Model-Vide-Controller. Цей шаблон передбачає поділ системи на три взаємопов’язані частини: модель даних, вигляд (інтерфейс користувача) та модуль

керування. Застосовується для відокремлення даних (моделі) від інтерфейсу користувача (вигляду) так, щоб зміни інтерфейсу користувача мінімально впливали на роботу з даними, а зміни в моделі даних могли здійснюватися без змін інтерфейсу користувача. Мета шаблону — гнучкий дизайн програмного забезпечення, який повинен полегшувати подальші зміни чи розширення програм, а також надавати можливість повторного використання окремих компонентів програми. Крім того використання цього шаблону у великих системах сприяє впорядкованості їхньої структури і робить їх більш зрозумілими за рахунок зменшення складності.

На рисунку 3.4 є такі позначення що потребують пояснень:

– пакет «Model» містить моделі даних, відповідно пакети «View» та «Controller» містять класи інтерфейсу користувача та контролери;

– інтерфейс IManifestable та абстрактний клас Mainfest надають можливість системі отримувати інформацію про доступні model, view та conreoller’и без власне завантаження їх в пам’ять програми; для отримання інформації про наявні компонент достатньо лише завантажити відповідні маніфести;

– класи GenericView, GenericModel, GenericController надають програмісту ”скелет” готової компоненти при розробці нових, таким чином програмісту не потрібно кожного разу імплементувати однакові методи, які, можливо, вимагає ядро системи.

Концептуальна модель даних. Модель даних – абстрактне представлення реального світу, що відображає тільки ті об’єкти, що безпосередньо стосуються програми. Це, як правило, визначає специфічну групу об’єктів, їх атрибутивне значення і відношення між ними [19].

Після аналізу предметної області було запропоновано таку концептуальну модель даних в нотації UML, як зображено на рисунку 3.5.

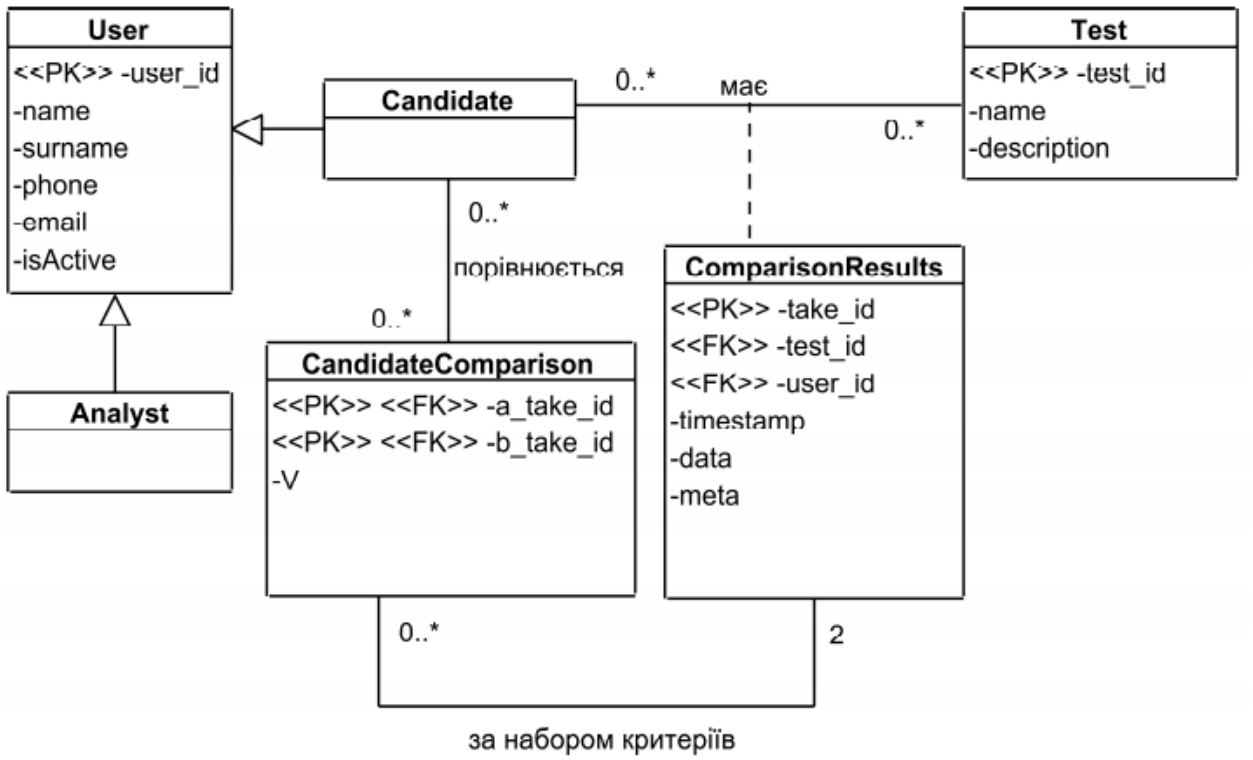

Рисунок 3.5 – Концептуальна модель даних програмного забезпечення роботи в нотації UML

На діаграмі передбачені сутності для зберігання інформації про користувачів (сутність «User») та про тести (сутність «Test»), які вони можуть проходити декілька разів. Також передбачено зберігання результатів порівняння отриманих показників (сутності «CandidateComparison» та «ComparisonResults»).

3.4 Вибір інструментальних засобів реалізації програмного забезпечення

На сьогоднішній день ринок надає широкий вибір технологій реалізації програмного забезпечення роботи. В якості потенційних засобів розробки програмного забезпечення для дослідження операцій розглядалися наступні технології: Java, Python, ASP.NET, PHP. Список з цих технологій був складений

виходячи зі знань автора кожної з них, що дозволяє виключити з плану роботи витрати часу на вивчення нової технології.

При виборі інструментарію були сформульовані відповідні критерії:

– крос-платформеність – один і той же програмний код повинен покрити якомога більше платформ;

– доступність – розробка повинна бути максимально дешевою;

– простота розробки – час розробки має бути мінімальним;

– наявність зручного та сучасного GUI-фреймворку для швидкої розробки графічного інтерфейсу;

– наявність простих інструментів статичного аналізу коду.

Java – об’єктно-орієнтована мова програмування, випущена 1995 року компанією «Sun Microsystems» як основний компонент платформи Java. З 2009 року мовою займається компанія «Oracle», яка того року придбала «Sun Microsystems». В офіційній реалізації Java-програми компілюються у байт-код, який при виконанні інтерпретується віртуальною машиною для конкретної платформи.

«Oracle» надає компілятор Java та віртуальну машину Java, які задовольняють специфікації Java Community Process, під ліцензією GNU General Public License.

Мова значно запозичила синтаксис із C і C++. Зокрема, взято за основу об’єктну модель C++, проте її модифіковано. Усунуто можливість появи деяких конфліктних ситуацій, що могли виникнути через помилки програміста та полегшено сам процес розробки об’єктно-орієнтованих програм. Ряд дій, які в C/C++ повинні здійснювати програмісти, доручено віртуальній машині.

Передусім Java розроблялась як платформо-незалежна мова, тому вона має менше низькорівневих можливостей для роботи з апаратним забезпеченням, що в порівнянні, наприклад, з C++ зменшує швидкість роботи програм. За необхідності таких дій Java дозволяє викликати підпрограми, написані іншими мовами програмування.

Java вплинула на розвиток J++, що розроблялась компанією «Microsoft». Роботу над J++ було зупинено через судовий позов «Sun Microsystems», оскільки ця мова програмування була модифікацією Java. Пізніше в новій платформі «Microsoft» .NET випустили J#, щоб полегшити міграцію програмістів J++ або Java на нову платформу. З часом нова мова програмування C# стала основною мовою платформи, перейнявши багато чого з Java. J# востаннє включався в версію Microsoft Visual Studio 2005.

ASP.NET — технологія створення веб-застосування і веб-сервісів від компанії Майкрософт. Вона є складовою частиною платформи Microsoft.NET і розвитком старішої технології Microsoft ASP. У цей час останньою версією цієї технології є ASP.NET 5.

ASP.NET зовні багато в чому зберігає схожість із старішою технологією ASP, що дозволяє розробникам відносно легко перейти на ASP.NET. У той же час внутрішній устрій ASP.NET істотно відрізняється від ASP, оскільки вона заснована на платформі. NET і, отже, використовує всі нові можливості, що надаються цією платформою.

Microsoft повністю перебудувала ASP.NET, ґрунтуючись на Common Language Runtime (CLR), який є основою всіх застосунків Microsoft .NET. Розробники можуть писати код для ASP.NET, використовуючи практично будь-які мови програмування, що входять у комплект .NET Framework (C#, Visual Basic.NET, і JScript.NET). ASP.NET має перевагу у швидкості в порівнянні зі скриптовими технологіями, тому що при першому зверненні код компілюється і поміщається в спеціальний кеш, і згодом тільки виконується, не вимагаючи витрат часу на парсинг, оптимізацію ті інші проміжні операції.

Python – інтерпретована об’єктно-орієнтована мова програмування високого рівня з строгою динамічною типізацією. Структури даних високого рівня разом із динамічною семантикою та динамічним зв’язуванням роблять її привабливою для швидкої розробки програм, а також як засіб поєднання існуючих компонентів. Python підтримує модулі та пакети модулів, що сприяє модульності та повторному використанню коду. Інтерпретатор Python та стандартні бібліотеки

доступні як у скомпільованій так і у вихідній формі на всіх основних платформах. В мові програмування Python підтримується декілька парадигм програмування, зокрема: об’єктно-орієнтована, процедурна, функціональна та аспектно-орієнтована.

Серед основних її переваг можна назвати такі:

– чистий синтаксис (для виділення блоків слід використовувати відступи);

– переносність програм (що властиве більшості інтерпретованих мов);

– можливість використання Python в діалоговому режимі;

– стандартний дистрибутив має просте, але разом із тим досить потужне середовище розробки, яке зветься IDLE і яке написане на мові Python;

– зручний для розв’язання математичних проблем (має засоби роботи з комплексними числами, може оперувати з цілими числами довільної величини, у діалоговому режимі може використовуватися як потужний калькулятор);

– відкритий код.

Виділяють такі недоліки Python:

– Низька швидкодія;

– Відсутність статичної типізації;

– Неможливість модифікації вбудованих класів (int, str та інших подібних).

PHP – скриптова мова програмування, була створена для генерації HTMLсторінок на стороні веб-сервера. PHP є однією з найпоширеніших мов, що використовуються у сфері веб-розробок (разом із Java, .NET, Perl, Python, Ruby).

PHP підтримується переважною більшістю хостинг-провайдерів. PHP – проект відкритого програмного забезпечення. PHP інтерпретується веб-сервером у HTML-код, який передається на сторону клієнта. На відміну від скриптової мови JavaScript, користувач не бачить PHP- коду, бо браузер отримує готовий html-код. Це є перевага з точки зору безпеки, але погіршує інтерактивність сторінок. Але ніхто не забороняє використовувати PHP для генерування JavaScript-кодів, які виконуються вже на стороні клієнта.

Серед основних переваг PHP можна назвати такі:

– PHP – С-подібна мова програмування, тому синтаксис буде здаватися знайомим;

– спрямованість на веб-розробку: вузький круг задач, що вирішуються, не вимагає від програміста надмірного вивчення додаткових технологій;

– висока популярність;

– вільна ліцензія.

Виділяють такі недоліки PHP:

– змінні з символом ”$”;

– складні назви поширених функцій (htmlspecialchars, mysql_select_db, nl2br тощо);

– не підтримується Unicode в версіях до 6.0.

Порівняння технологій за обраними критеріями наведені в таблиці 3.1.

Таблиця 3.1 – Порівняння технологій реалізації

Назва	Крос-платформеність	Доступність	Простота розробки	GUI фреймворки	Інструменти статистичного аналізу
Java	Виконується у віртуальній машині. Реалізовано механізм just in time compilation. Віртуальна машина доступна на більшості ОС	Ліцензія вільна, вихідний код доступний; IDE, компілятор і віртуальна машина безкоштовні	Сувора типізація, орієнтація на ООП	Swing	PMD, Soot та інші
ASP.NET	Виконується у віртуальній машині. Інтерпретується, компілюється в байт-код. Віртуальна машина доступна на Windows, частково на Linux (Mono)	Ліцензія не вільна, вихідний код доступний; IDE, компілятор і віртуальна машина пропрієтарні, але безкоштовні	Сувора типізація, орієнтація на ООП	Windows Forms, WPF	PVSStudio, Nitriq та близько 10 інших

Закінчення таблиці 3.1

Python	Виконується у віртуальній машині. Інтерпретується, компілюється в байт-код. Віртуальна машина доступна на більшості ОС	Ліцензія вільна, вихідний код доступний; IDE, компілятор і віртуальна машина безкоштовні	Динамічна типізація, відсутність орієнтації на певну парадигму	QT	pylint, pyflakers, mypy
PHP	Виконується у віртуальній машині. Інтерпретується, компілюється в байт-код. Віртуальна машина доступна на більшості ОС	Ліцензія вільна, вихідний код доступний; IDE, інтерпретатор безкоштовні	Динамічна типізація, ООП парадигма	HTML	PHP Code Sniffer – добре знайомий автору фреймворк статичного аналізу

На сьогоднішній день ринок надає широкий вибір технологій реалізації програмного забезпечення роботи. В якості потенційних засобів розробки програмного забезпечення для дослідження операцій розглядалися наступні технології: Java, Python, ASP.NET, PHP. Список з цих технологій був складений виходячи зі знань автора кожної з них, що дозволяє виключити з плану роботи витрати часу на вивчення нової технології.

Після аналізу зазначених технологій, автор прийшов до висновку про необхідність використання PHP як єдиного інструменту розробки: автор на момент початку роботи вже був добре знайомий з цією технологією та інструментом статичного аналізу програмного коду PHP Code Sniffer, мова програмування PHP дещо схожа на Java (через те що має C-подібний синтаксис) тому імплементація метрик авторства коду не буде потребувати їх адаптації.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ЗАПРОПОНОВАНОГО ПІДХОДУ

4.1 Вимоги до програмного забезпечення

Програмне забезпечення складається з декількох частин, кожна з яких має своє призначення, свою вхідну та вихідну інформацію:

1) Сервер накопичення статистичних даних збирає відповіді респондентів за тестом KTS та зразки програмного коду. Таким чином вхідною інформацією можна вважати відповіді респондентів, а вихідною – записи в СКБД, що містять інформацію про відповіді користувачів а також деякі проміжні значення.

2) Сервер аналізу користувальницького коду розроблено для валідації та статичного аналізу коду, таким чином вхідною інформацією виступають зразки програмного коду, що автоматично звантажуються з серверу накопичення даних через власний API. Після ізоляції користувальницького коду у віртуальному середовищі, юніт-тестування та статичного аналізу, статистичні дані вивантажуються у відповідну СКБД. Такі дані і є вихідною інформацією системи.

Після кожної ітерації процесу збору статистичних даних та аналізу зразків програмного коду, аналітик отримує дані, що дозволяють оцінити наявність кореляції між особистими якостями респондента та програмним кодом, що він розробив.

Висуваються такі вимоги до апаратного забезпечення користувача:

– процесор Intel Pentium 4 або будь-який інший сумісний з підтримкою набору інструкцій SSE2;

– RAM 1024 МБ і більше;

– HDD 512 МБ і більше.

Вимоги до програмного забезпечення користувача:

– операційна система Windows 7 або Windows 8, або Windows 8.1, або

Windows 10, або OS X Mavericks 10.9 та новіші, або 64-bit Ubuntu 14.04 та новіші, або Debian 8 та новіші, або openSUSE 13.3 та новіші, або Fedora Linux 24 та новіші;

– браузер Google Chrome 58.0 та новіші, або будь-який інший сумісний.

Оскільки програмне забезпечення роботи розроблено з урахуванням особливостей клієнт-серверної архітектури, до програмного забезпечення серверу аналізу користувальницького коду висуваються додаткові вимоги щодо програмного забезпечення:

– Oracle VM Virtual Box 5.0 та новіше, або будь-якої іншої сумісної системи віртуалізації;

– підтримка системи управління віртуалізацією Vagrant 1.9.5 та новіше;

– СКБД MySQL 5 та новіше, або будь-яка інша сумісна.

Окремо виділяються вимоги щодо програмного забезпечення порталу збору користувальницьких даних:

– веб-сервер Apache 2 та новіше, або будь-який інший сумісний;

– PHP 5 та новіше;

– СКБД MySQL 5 та новіше, або будь-яка інша сумісна.

4.2 Тестування системи

Планування робіт. Згідно до стандарту IEEE 829-1998 [20], тест-план – це документ, що описує масштаби, підхід, ресурси і графік тестування. Він визначає тестові завдання, особливості, які будуть перевірені, завдання тестування, відповідальних осіб, які будуть робити кожну задачу, і будь-які ризики, що вимагають планування на випадок непередбачених обставин.

Однією з найбільш критичних частин програмного забезпечення роботи є імплементація метрик авторства програмного коду, що були програмно імплементовані на базі фреймворка PHPCS. З оглядом на обмежений час,

виділений на тестування програмного забезпечення, тестування було проведене тільки для вказаних частин ПЗ.

Тест-план має таку структуру:

1) унікальна назва тест-плану;

2) вступ;

3) артефакти для тестування, з включенням версій;

4) список функцій ПЗ для тестування;

5) список функцій ПЗ, тестування яких необхідно уникнути;

6) список основних заходів, методів та інструментів, які використовуються для тестування позначених груп функцій;

7) список критеріїв успіху або невдачі;

8) список критеріїв призупинення та відновлення тестування;

9) список результатів тестування: логи тестування, вихідні дані, звіти;

10) список завдань для підготовки тестування;

11) список вимог до середовища тестування;

12) список відповідальних осіб;

13) вимоги до рівня кваліфікації тестувальників;

14) розклад тестування;

15) ризики та план дій в надзвичайних ситуаціях;

16) додатки.

За цим шаблоном розроблено такий тест-план:

1 Назва: «Тестування імплементації метрик авторства програмного коду»;

2 Цей тест-план призначено для тестування модулів, що імплементують метрики авторства програмного коду для фреймворку статичного аналізу програмного коду PHPCS. Тестування перевіряє загальну правильність роботи модулів;

3 Артефакти для тестування, з включенням версій;

а) модуль ”SLOC”, версія 6596f88;

б) модуль ”PRO2b”, версія 6596f88;

в) модуль ”STY1c”, версія 6596f88;

г) модуль ”STY1g”, версія 6596f88;

д) модуль ”STY1h”, версія 6596f88;

е) модуль ”STY2a”, версія 6596f88;

є) модуль ”STY5”, версія 6596f88;

4 Список функцій ПЗ для тестування;

а) вимірювання показника SLOC;

б) вимірювання показника PRO2b;

в) вимірювання показника STY1c;

г) вимірювання показника STY1g;

д) вимірювання показника STY1h;

е) вимірювання показника STY2a;

є) вимірювання показника STY5;

5 Уникнути тестування службових функції фреймворку PHPCS;

6 Список основних заходів, методів та інструментів, які використовуються для тестування позначених груп функцій;

а) рівень тестування: модульне (unit-тестування);

б) інструменти: мова програмування PHP, фреймворк юніт-тестування PHPUnit;

в) Виконання користувальницького програмного коду не допускається.

7 Список критеріїв успіху або невдачі. Визнати невдачу, якщо хоча б одна з умов не виконується;

а) показники ”STY*”, ”SLOC” та ”PRO*” набувають відповідних до поданого тестового програмного коду значень;

б) аналіз програмного коду в коментарях не відбувається;

8 Список критеріїв призупинення та відновлення тестування:

а) призупинити тестування у випадку дефекту фреймворка PHPCS до його виправлення;

9 Список результатів тестування;

а) звіт інструменту «Test Explorer» зі статусом unit-тестування;

б) «Coverage Report» про рівень покриття юніт-тестами коду модулів.

Проектування і виконання тестів. Для проектування та власне розробки тестів використано фреймворк PHPUnit. Цей фреймворк декларує єдиний вид всіх тестів, приклад наведено нижче.

```
<?php

namespace MyApplication \ Tests;

use PHPUnit_Framework_TestCase;
use PHP_CodeSniffer_CLI;

class TestCase extends PHPUnit_Framework_TestCase
{
    Public function testName()
    {
        $this->assertTrue(false);
    }
}
```

Фреймворк надає декілька методів, що спрощують розробку, серед них ”assertTrue”, ”assertEmpty” та інші. Кожен з методів класу ім’я якого закінчується на ”Test” є юніт-тестом.

Вихідний код unit-тестів наведено на рисунках Б.1 – Б.2 в додатку Б.

Аналіз результатів тестування. Результат тестування модуля SLOC наведено в таблиці 4.1.

Таблиця 4.1 – Тест «testSLOC»

Показник	Значення
Опис тексту	Перевіряється коректність вимірювання показника SLOC
Початкові умови	Модуль SLOCSniff завантажується фреймворком PHPCS
Очікуваний результат	Значення показника відповідає дійсному значенню, актуальному для поданого для аналізу зразка коду
Результат тесту	Тест пройдений п’ять разів – помилок не виявлено

Результат тестування модуля PRO2b наведено в таблиці 4.2.

Таблиця 4.2 – Тест «testPRO2b»

Показник	Значення
Опис тексту	Перевіряється коректність вимірювання показника PRO2b
Початкові умови	Модуль PRO2bSniff завантажується фреймворком PHPCS
Очікуваний результат	Значення показника відповідає дійсному значенню, актуальному для поданого для аналізу зразка коду
Результат тесту	Тест пройдений п’ять разів – помилок не виявлено

Результат тестування модуля STY1c наведено в таблиці 4.3.

Таблиця 4.3 – Тест «testSTY1c»

Показник	Значення
Опис тексту	Перевіряється коректність вимірювання показника STY1c
Початкові умови	Модуль STY1cSniff завантажується фреймворком PHPCS
Очікуваний результат	Значення показника відповідає дійсному значенню, актуальному для поданого для аналізу зразка коду
Результат тесту	Тест пройдений п’ять разів – помилок не виявлено

Результат тестування модуля STY1g наведено в таблиці 4.4.

Таблиця 4.4 – Тест «testSTY1g»

Показник	Значення
Опис тексту	Перевіряється коректність вимірювання показника STY1g
Початкові умови	Модуль STY1gSniff завантажується фреймворком PHPCS
Очікуваний результат	Значення показника відповідає дійсному значенню, актуальному для поданого для аналізу зразка коду
Результат тесту	Тест пройдений п’ять разів – помилок не виявлено

Результат тестування модуля STY1h наведено в таблиці 4.5.

Таблиця 4.5 – Тест «testSTY1h»

Показник	Значення
Опис тексту	Перевіряється коректність вимірювання показника STY1h
Початкові умови	Модуль STY1hSniff завантажується фреймворком PHPCS
Очікуваний результат	Значення показника відповідає дійсному значенню, актуальному для поданого для аналізу зразка коду
Результат тесту	Тест пройдений п’ять разів – помилок не виявлено

Результат тестування модуля STY2a наведено в таблиці 4.6.

Таблиця 4.6 – Тест «testSTY2a»

Показник	Значення
Опис тексту	Перевіряється коректність вимірювання показника STY2a
Початкові умови	Модуль STY2aSniff завантажується фреймворком PHPCS
Очікуваний результат	Значення показника відповідає дійсному значенню, актуальному для поданого для аналізу зразка коду
Результат тесту	Тест пройдений п’ять разів – помилок не виявлено

Результат тестування модуля STY5 наведено в таблиці 4.7.

Таблиця 4.7 – Тест «testSTY5»

Показник	Значення
Опис тексту	Перевіряється коректність вимірювання показника STY5
Початкові умови	Модуль STY5Sniff завантажується фреймворком PHPCS
Очікуваний результат	Значення показника відповідає дійсному значенню, актуальному для поданого для аналізу зразка коду
Результат тесту	Тест пройдений п’ять разів – помилок не виявлено

За результатми тестування консольною утілитою phpunit був складений «Coverage report», який наведено на рисунках 4.1 – 4.1.

Звіт на рисунках 4.3 – 4.4 свідчить про достатній рівень покриття тестами програмного коду, що вимагає тестування. Також тестування дещо торкнулося службового коду фреймворку PHPCS.

Процес інсталяції ПЗ. Користувачі, що надають статистичні дані, не потребують інсталяції будьякого додаткового програмного забезпечення. Портал для користувачів доступний через інтернет за адресою http://cw51.kostenko.name. Процес інсталяції серверного програмного забезпечення для розгортання порталу збору статистичних даних такий:

```
Code Coverage Report:
  2017-06-03 20:00:25

 Summary:
  Classes: 25.00% (4/16)
  Methods: 20.83% (35/168)
  Lines:   28.18% (1588/5635)

@Authorship.Sniffs.Metrics::Authorship_Sniffs_Metrics_PRO2bSniff
  Methods: 100.00% ( 3/ 3)   Lines: 100.00% ( 29/ 29)
@Authorship.Sniffs.Metrics::Authorship_Sniffs_Metrics_SLOCSniff
  Methods: 100.00% ( 3/ 3)   Lines: 100.00% ( 21/ 21)
@Authorship.Sniffs.Metrics::Authorship_Sniffs_Metrics_STY1cSniff
  Methods: 100.00% ( 3/ 3)   Lines: 100.00% ( 36/ 36)
```

Рисунок 4.1 – «Coverage report», складений за результатами тестування

```
@Authorship.Sniffs.Metrics::Authorship_Sniffs_Metrics_STY1gSniff
  Methods: 100.00% ( 1/ 1)   Lines: 100.00% (  1/  1)
@Authorship.Sniffs.Metrics::Authorship_Sniffs_Metrics_STY1hSniff
  Methods:  75.00% ( 3/ 4)   Lines:  97.50% ( 39/ 40)
@Authorship.Sniffs.Metrics::Authorship_Sniffs_Metrics_STY2aSniff
  Methods:  66.67% ( 2/ 3)   Lines:  96.30% ( 52/ 54)
@Authorship.Sniffs.Metrics::Authorship_Sniffs_Metrics_STY5Sniff
  Methods:  75.00% ( 3/ 4)   Lines:  97.44% ( 38/ 39)
@PHP_CodeSniffer::PHP_CodeSniffer
  Methods:  13.95% ( 6/43)   Lines:  28.31% (327/1155)
@PHP_CodeSniffer::PHP_CodeSniffer_CLI
  Methods:   0.00% ( 0/19)   Lines:  16.65% (138/829)
@PHP_CodeSniffer::PHP_CodeSniffer_File
  Methods:  19.23% (10/52)   Lines:  26.92% (498/1850)
@PHP_CodeSniffer::PHP_CodeSniffer_Fixer
  Methods:   0.00% ( 0/16)   Lines:   3.18% ( 10/314)
@PHP_CodeSniffer::PHP_CodeSniffer_Reporting
  Methods:   0.00% ( 0/ 6)   Lines:  58.47% (107/183)
@PHP_CodeSniffer::PHP_CodeSniffer_Reports_Json
  Methods:  50.00% ( 1/ 2)   Lines:  94.59% ( 35/ 37)
@PHP_CodeSniffer::PHP_CodeSniffer_Tokenizers_PHP
  Methods:   0.00% ( 0/ 4)   Lines:  28.81% (257/892)
```

Рисунок 4.2 – «Coverage report», складений за результатами тестування

Звіт на рисунках 4.3 – 4.4 свідчить про достатній рівень покриття тестами програмного коду, що вимагає тестування. Також тестування дещо торкнулося службового коду фреймворку PHPCS.

Процес інсталяції ПЗ. Користувачі, що надають статистичні дані, не потребують інсталяції будьякого додаткового програмного забезпечення. Портал для користувачів доступний через інтернет за адресою http://cw51.kostenko.name. Процес інсталяції серверного програмного забезпечення для розгортання порталу збору статистичних даних такий:

– створити директорію на сервері, що доступний мережею інтернет, директорія повинна бути відкрита для HTTP-серверу Apache;

– перейти в створену директорію;

– виконати команду gitclonehttps://github.com/Livich/master−diploma−www.git;

– виконати команду . /setup.sh.

Процес інсталяції серверного програмного забезпечення для аналізу користувальницького коду такий:

– створити директорію на сервері;

– перейти в створену директорію;

– виконати команду git clone https://github.com/Livich/master−diploma−code−sn iffer.git.;

– виконати команду vagrant up;

– виконати команду vag rant ssh , що відкриє віддалений термінал віртуальної машини;

– в віддаленому терміналі відкрити на редагування файл i ndex.php та користуючись інструкціями встановити значення API_DB_DUMP_ENDPOINT , API_DB_DUMP_ADDR, PDO_CONNECTION_STRING , PDO_USER, PDO_PASS;

– в віддаленому терміналі можна запускати аналіз вручну за допомогою команди . /vendor/squizlabs/php_codesniffer/scripts/phpcs−−standard=Authorship−−report=jsonfilename.php або запустивши скріпт index.php.

4.3 Опис візуального інтерфейсу системи

Програмне забезпечення роботи складається з декількох частин, але власний графічний інтерфейс має лише портал збору та накопичення статистичних даних. Інші програмні компоненти призначені лише для внутрішнього використання кваліфікованими користувачами, тому мають лише консольний інтерфейс.

Інтерфейс екрану реєстрації нового користувача для надання статистичних даних наведено на рисунку 4.3.

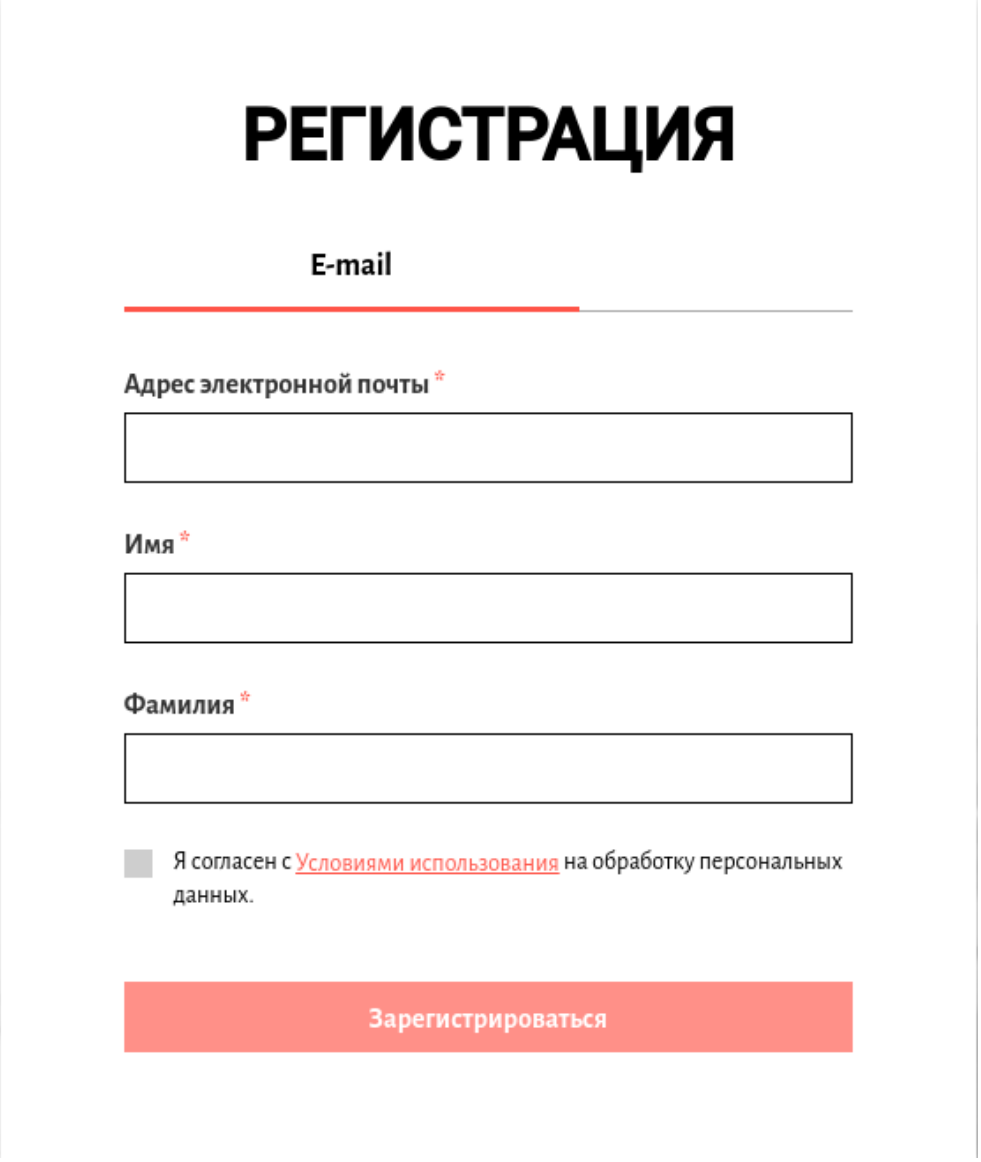

Рисунок 4.3 – Графічний інтерфейс екрану реєстрації нового користувача

На рисунку 4.3 позначені:

1) Поле вводу електронної адреси користувача. Поле має валідацію вводу.

2) Поле для вводу ім’я користувача.

3) Поле для вводу фамілії користувача (не обов’язково).

4) Поле для вводу пароля нового аккаунта.

5) Кнопка продовження процесу реєстрації.

6) Кнопка меню, що надає можливість відміни реєстрації.

7) Кнопка переходу до форми входу для користувачів, що вже зареєстровані в системі.

Графічний інтерфейс екрану входу користувача наведено на рисунку 4.4.

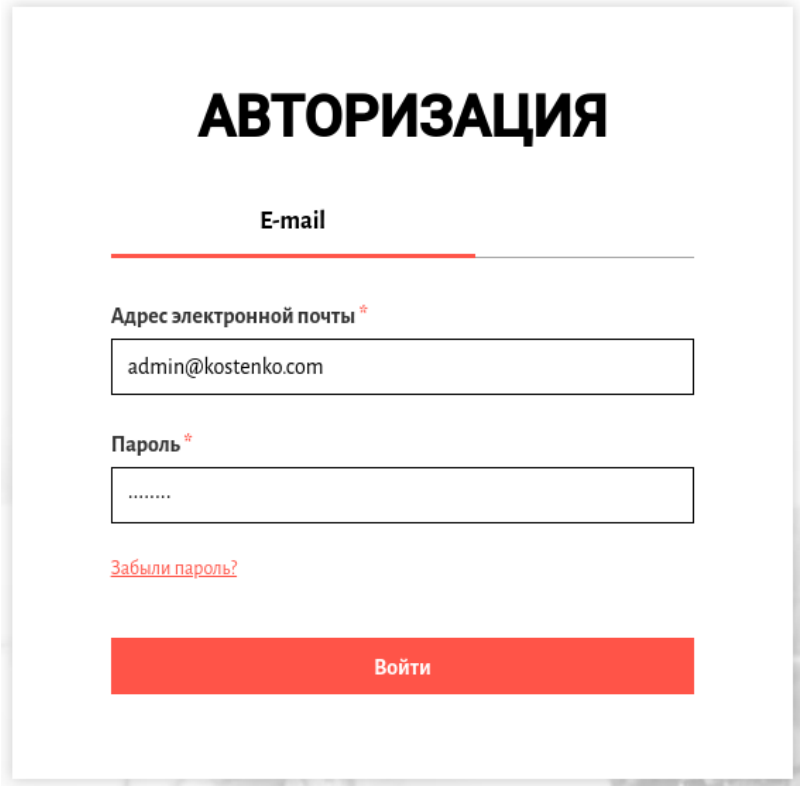

Рисунок 4.4 – Графічний інтерфейс екрану входу користувача

Н рисунку 4.4 позначені:

1) Поле вводу електронної адреси користувача. Поле має валідацію вводу.

2) Поле вводу пароля користувача, який він зберіг раніше.

3) Кнопка входу в систему.

4) Кнопка, що відкриває форму реєстрації.

5) Кнопка, що відкриває форму відновлення пароля.

6) Логотип, що веде на головну сторінку порталу.

7) Кнопка меню, що надає можливість відміни входу.

Графічний інтерфейс головної сторінки порталу збору статистики наведено на рисунку 4.5.

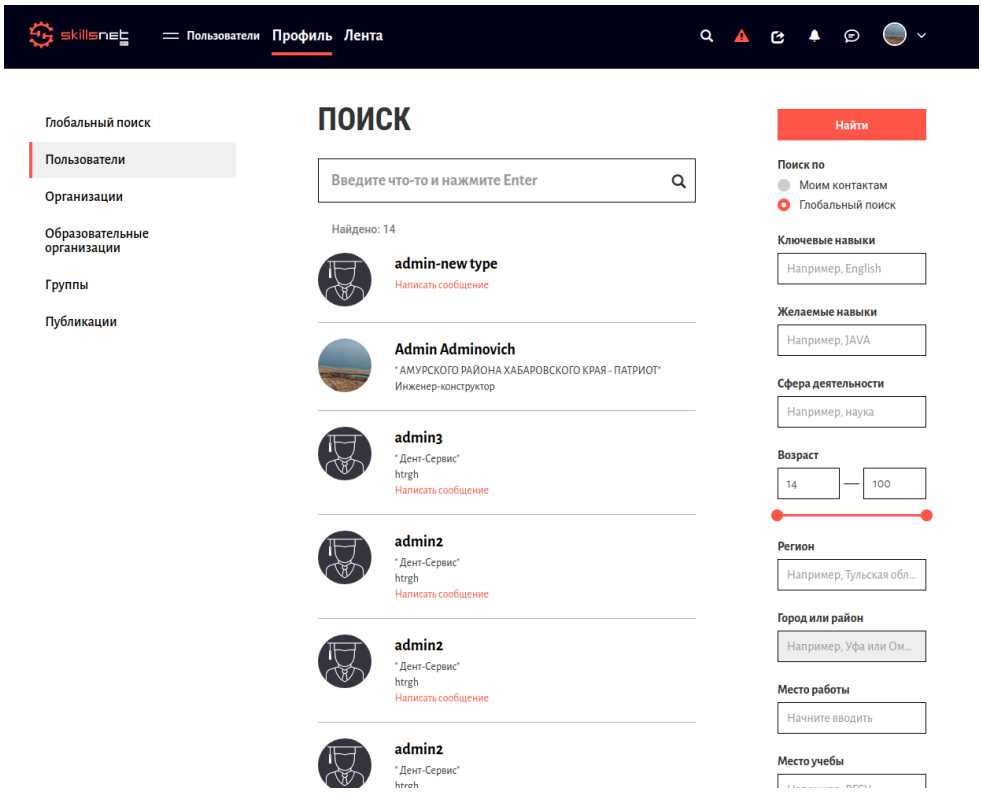

Рисунок 4.5 – Графічний інтерфейс головного екрану порталу збору та накопичення статистичних даних

На даному екрані користувачі системи можуть здійснювати пошук розробників, використовуючи різноманітні фільтри, такі як навички, вік, бажаний регіон, місце роботи або навчання та інші.

4.4 Апробація результатів роботи

В результаті роботи запропоновано метод оцінки суб’єктної диверсифікації на основі профілю розробника. Запропонований метод вимагає статистичного підтвердження, тож була розроблена програмна архітектура для засобу оцінки за запропонованим методом, а також веб-застосунок для збору та накопичення статистичних даних від респондентів.

За допомогою мови моделювання UML побудовано ряд діаграм, що формалізують запропоновану архітектуру програмних засобів. На їх основі розроблено працездатний прототип, що, використовуючи наявні вхідні дані у вигляді відповідей респондентів формує множину оцінок профілю розробника та множину показників метрик авторства наданого респондентом програмного коду. На закінчення показані перспективи роботи, сформульовано план майбутніх робіт.

Протягом всього часу дослідження результати були представлені в рамках 3-ї міжнародної науково-технічної конференції «Комп’ютерні та інформаційні системи і технології» [21].

ВИСНОВКИ

У ході роботі розглянуто питання формування команд розробників програмного забезпечення за допомогою професійних соціальних мереж. Така модель дозволяє формувати команду на основі даних, які знаходяться в соціальній мережі, таких як досвід, навички, відгуки тощо. Керуючись цими та подібними атрибутами система виконує розподіл розробників за командами. Дана робота надає метрики та модель для оцінки рівня суб'єктної диверсифікації спираючись на атрибути розробника, та на атрибути програмного коду що він виробляє. Висувається гіпотеза про зв'язок між профілем особистості розробника та набором певних атрибутів його професійних якостей.

Розглянуті основні проблеми початкових етапів формування команд розробників: недостатня формалізованість процесу формування команд розробників, розрізненість існуючих метрик, що використовуються в цьому процесі. Встановлено, що зазначені недоліки призводять до ускладнення процесу отримання оптимального складу команд розробників програмного забезпечення: розглянуто декілька моделей формування команд, які спираються на різнорідні метрики, такі як вік, стать, розуміння розробниками цілей проекту, тощо.

Разом із тим проведено огляд деяких методів оцінки рівня формування команд розробників. Виявлено, що подібні метрики спираються на безпосередні результати одночасного тестування версій на наборах вхідних даних. Отже, прямого зв'язку між методом формування команд та показником досягнутої диверсифікації не знайдено, це дає можливість припускати що зміна методу формування команд розробників не дає безпосередьного впливу на показник диверсифікації, що дозволяє використовувати останній як метрику ефективності запропонованого в роботі методу.

Для досягнення мети застосовувались такі методи: Knowledge Diversity (для отримання індексу на основі показника ентропії); (Social category Diversity (для

дослідження впливу впливу соціальної категорії); Value Diversity (для оцінки ставлення кандидата до того чи іншого питання); методи оцінки типу особистості та інші.

В результаті роботи запропоновано метод оцінки суб’єктної диверсифікації на основі профілю розробника. Запропонований метод вимагає статистичного підтвердження, тож була розроблена програмна архітектура для засобу оцінки за запропонованим методом, а також веб-застосунок для збору та накопичення статистичних даних від респондентів.

Знайдено засіб, який би дозволив виявляти певний набір якостей деяких кандидатів, отримано оцінки за певним набором якостей на певній множині кандидатів, згідно до висунутої гіпотези дослідження, отримано оцінки складності програмного продукту, що потребує, оцінено кореляцію між рівнем складності програмного продукту та між оцінками за бажаним набором якостей кандидатів.

За допомогою мови моделювання UML побудовано ряд діаграм, що формалізують запропоновану архітектуру програмних засобів. На їх основі розроблено працездатний прототип, що, використовуючи наявні вхідні дані у вигляді відповідей респондентів формує множину оцінок профілю розробника та множину показників метрик авторства наданого респондентом програмного коду. На закінчення показані перспективи роботи, сформульовано план майбутніх робіт.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1 Ian Sommerville. Software Engineering. — Pearson Education, 2007. — ISBN: 9780321313799.

2 Lyu Michael R, He Y-T. Improving the n-version programming process through the evolution of a design paradigm // IEEE Transactions on Reliability. — 1993. — Vol. 42, no. 2. — P. 179–189.

3 Avizienis Algirdas, Chen Liming. On the implementation of N-version programming for software fault tolerance during execution // Proc. IEEE COMPSAC. – – Vol. 77. — 1977. — P. 149–155.

4 The impact of value diversity on information system development projects / TingPeng Liang, Jason Chia-Hsien Wu, James J Jiang, Gary Klein // International Journal of Project Management. — 2012. — Vol. 30, no. 6. — P. 731–739.

5 Baudry Benoit, Monperrus Martin. The multiple facets of software diversity: Recent developments in year 2000 and beyond // ACM Computing Surveys (CSUR). — 2015. — Vol. 48, no. 1. — P. 16.

6 Effect of team diversity on software project performance / Ting-Peng Liang, Chih-Chung Liu, Tse-Min Lin, Binshan Lin // Industrial Management & Data Systems. — 2007. — Vol. 107, no. 5. — P. 636–653.

7 Jehn Karen A, Northcraft Gregory B, Neale Margaret A. Why differences make a difference: A field study of diversity, conflict and performance in workgroups // Administrative science quarterly. — 1999. — Vol. 44, no. 4. — P. 741–763.

8 Teachman Jay D. Analysis of population diversity measures of qualitative variation // Sociological Methods & Research. — 1980. — Vol. 8, no. 3. — P. 341–362.

9 O’Reilly Charles A, Chatman Jennifer, Caldwell David F. People and organizational culture: A profile comparison approach to assessing person-organization fit //Academy of management journal. — 1991. — Vol. 34, no. 3. — P. 487–516.

10 Lüscher Color Diagnostik. — URL: http://www.luscher-color.com/ (online; accessed: 04/06/2017).

11 Mark J. Hilsenroth, Daniel L. Segal, Michel Hersen. Comprehensive Handbook of Psychological Assessment, Personality Assessment (Comprehensive Handbook of Psychological Assessment). — 1 edition. — Wiley, 2003. — Vol. 2. — ISBN: 9780471416128.

12 Isabel Briggs Myers. Introduction to Type: A Guide to Understanding Your Results on the MBTI Instrument. — Cpp, Inc., 1998. — ISBN: 0050436074.

13 David Keirsey, Marilyn M. Bates. Please Understand Me. — Prometheus Nemesis, 1984. — ISBN: 9780960695409.

14 Ding Haibiao, Samadzadeh Mansur H. Extraction of Java program fingerprints for software authorship identification // Journal of Systems and Software. — 2004. — Vol. 72, no. 1. — P. 49–57.

15 Littlewood B., Miller D. R. Conceptual Modeling of Coincident Failures in Multiversion Software // IEEE Trans. Softw. Eng. — 1989. — Vol. 15, no. 12. — P. 1596– 1614. — URL: http://dx.doi.org/10.1109/32.58771.

16 Partridge Derek, Krzanowski Wojtek. Distinct failure diversity in multiversion software // Res. Rep. — 1997. — Vol. 348. — P. 24.

17 Grady Booch, James Rumbaugh, Ivar Jacobson. The Unified Modeling Language User Guide (2nd Edition). — Addison-Wesley Professional, 2005. — ISBN: 0321267974.

18 Kruchten P. The Rational Unified Process: An Introduction. — Addison-Wesley Professional, 2003. — 336 p. — ISBN: 0321197704.

19 Thomas M. Connolly, Carolyn E. Begg. Database Systems: A Practical Approach to Design, Implementation and Management (5th Edition). — Pearson, 2009. — ISBN: 0321523067.

20 IEEE Standard for Software and System Test Documentation // IEEE Std 829-2008. — P. 150.

21 Kostenko O.M., A Human-Centered Approach to Assessment of Team Diversity in Multi-Version Software Development // Proceedings of the PhD Symposium at 13th International Conference on ICT in Education, Research, and Industrial Applications co-located with 3th International Conference on ICT in

Education, Research, and Industrial Applications / Ed. by Frédéric Mallet, Grygoriy Zholtkevych. — PhD Symposium at ICTERI 2017 no. 1. — 2019. — P. 18–21.

22 Лютенко, И.В., Лиликович, С.А. Измерение субъектной диверсности в условиях многоверсионной разработки программного обеспечения // SAIT 2016. Book of abstracts. — 2016. — С. 385.

23 Лілікович, С.О., Лютенко, І.В. Суб'єктно-орієнтований підхід до забезпечення диверсності N-версійного програмного проекту // SAIT 2017. Book of abstracts. — 2017. — С. 295.

24 Орловський, Д.Л., Лютенко, І.В., Лілікович, С.О. Підхід та засоби вимірювання суб'єктної диверсності в умовах багатоверсійної розробки програмного забезпечення // Вісник Національного Технічного Університету «ХПІ». — 2016. — Т. 1209, № 37.

25 Лілікович С.О. Розробка моделей та математичного забезпечення для вимірювання суб'єктної диверсності в умовах багатоверсійного виробництва ПЗ : КР / С.О. Лілікович ; Національний технічний університет «Харківський політехнічний інститут». — 2015.

26 Лілікович С.О. Підхід до вимірювання та проектування програмного забезпечення для оцінки суб'єктної диверсності в умовах багатоверсійної розробки програмних продуктів : КР / С.О. Лілікович ; Національний технічний університет «Харківський політехнічний інститут». — 2016.

27 Лілікович С.О. Розробка модельно-технологічного інструментарію для забезпечення суб'єктної диверсності в процесах створення багатоверсійних програмних продуктів : КР / С.О. Лілікович ; Національний технічний університет «Харківський політехнічний інститут». — 2017.