М. С. Пасєка, Н. М. Пасєка,
Ю. Л. Романишин, В. І. Шекета

ГРУПОВА ДИНАМІКА ЕФЕКТИВНИХ КОМАНД РОЗРОБНИКІВ

МОНОГРАФІЯ

м. Івано-Франківськ
2022

УДК 005.7
Г 90

Рецензенти:

Тимків Н. М. **–** д-р педаг. наук, професор, завідувачка кафедри англійської мови Івано-Франківського національного технічного університету нафти і газу

Яцків В. В. – д-р техн.. наук, доцент, завідувач кафедри кібербезпеки Західноукраїнського національного університету.

Алексєєв М. О. – д-р техн. наук, професор, декан факультету “Дніпровська політехніка”.

Пасєка М. С., Пасєка Н. М., Романишин Ю. Л., Шекета В. І.
Г 90 Групова динаміка ефективних команд розробників : монографія. Івано-Франківськ : ІФНТУНГ, 2022. 302 с.

*Рекомендовано до друку Вченою радою Івано-Франківського національного технічного університету нафти і газу (протокол **13/631** від **29 грудня 2021 р.**)*

ISBN **978–966–694–392–0**

В пропонованій монографії проведено системний огляд математичних моделей формування та функціонування членів команд розробників програмних систем з використанням хмарної технології, що відображають загальну методологію побудови та вивчення прикладних математичних моделей динамічного функціонування фірм – розробників програмних систем, а також можуть бути ефективно використані при вирішенні задач широкого класу управління соціально-економічними системами.

Ґрунтуючись на системному аналізі у досліджені групової динаміки для формування команд з використанням визначених особистісних профілів членів розробників програмних систем проведено логіко-функціональний аналіз адаптивних моделей тестування знань претендентів у команди розробників програмних систем. Розроблено математичні методи планування й опрацювання результатів адаптивного тестування набутих компетенцій, із можливістю статистичногоопрацювання даних.

Рекомендовано для *студентів та фахівців* в області програмних та інформаційних технологій.

УДК 005.7

ISBN **978–966–694–392–0**

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

ПЗ – програмне забезпечення
СКБД – Система керування базами даних
СКСД – Система керування сховищами даних
DNS – Domain Name System
HTML – HyperText Markup Language
IaaS – Infrastructure as a service
IT – Інформаційні технології
MSF – Microsoft Solutions Framework
NUMA – Non-Uniform Memory Access
POP – Post Office Protocol
RUP – Rational Unified Process
SLA – Service-level agreement
TCP – Transmission Control Protocol
UMA – Uniform memory access
ЖЦ – Життєвий цикл
ММО – Мережі масового обслуговування
ОВ – Обчислювальні вузли
ОМ – Обчислювальні машини
ОР – Обчислювальні ресурси
ОС – Операційна система
ПК – Персональний комп’ютер
ПП – Програмний продукт
ПС – Програмні системи
СБН – Сервіс балансування навантаження
ЦП – центральний процесор

ВСТУП

Актуальність теми. Сучасний стан інформаційного суспільства базується на нагальній необхідності збору, опрацювання й передачі значних обсягів потоків інформації та пов'язаний із нею, що створило передумови переходу суспільства від індустріального до інформаційного, або «Internet +». Глобальна інформатизація сучасного суспільства – це неперервний соціальний процес, в основі якого лежить провідна діяльність людства у сферах суспільного виробництва матеріальних благ, де продукування, збір, опрацювання, зберігання й, передача інформаційних потоків даних забезпечується засобами сучасної комп′ютерної обчислювальної техніки.

У сучасному інформаційному суспільстві основний акцент робиться на комплексі інноваційних заходів, спрямованих на швидке отримання та використання достовірних даних, які є вичерпні та своєчасні для галузей людської діяльності. Основною метою сучасної інформатизації суспільства є забезпечення виробничих процесів (бізнес-логіки), що сприяють не тільки лавиноподібному науково-технічному прогресу, а й створенню інноваційного інформаційного середовища для підтримки прийняття управлінських рішень. Основними критерієм інформатизації суспільства є створення програмно-технічних систем для швидкого опрацювання, передавання і зберігання надвеликих інформаційних потоків даних, зокрема використовуючи незалежні обчислювальні платформи.

У наукових публікаціях багато вчених звертають увагу на методи взаємодії команд, використовуючи групову динаміку та опису їх методів. Проте небагато вітчизняних та зарубіжних науковців звертає увагу в наукових публікаціях на аналіз впливу методів групової динаміки при формуванні команд розробників програмних систем із використанням хмарної технології. Ще одним з аспектів, на які менше приділяють увагу є аналіз впливу розміру і структури команди (гендерний підхід) за вибором реалізованих методик та їх вплив на досягнення позитивних результатів у розробленні програмних систем. Сьогодні багато авторів продовжують проводити практичні та наукові дослідження групової динаміки, зокрема енеаграм та сфер їх застосу-

вання. Серед відомих науковців варто перерахувати такі імена: Клаудіо Наранхо, Дон Річард Рісо і Рас Хадсон, А. Х. Алмаас, Річард Рор, Елізабет Вейгл, Джинджер Лапід-Богда, Хелена Макані.

Основним напрямом розвитку інформаційного суспільства в Україні є індустрія з розробки програмних систем на базі використання інноваційних конкурентоспроможних технологій, яка активно розвивається оскільки застосування інноваційних інформаційних технологій значно розширює сферу використання потоків даних у різних галузях завдяки модернізації технологічних процесів та підходів до характеру праці, зокрема зменшення співвідношення розумової праці до фізичної, мінімізації впливу людських помилок шляхом автоматизації бізнес-процесів.

Системний аналіз досліджень і публікацій та теоретичне підґрунтя опрацювання цієї проблеми професійного спілкування відображено у працях вітчизняних і зарубіжних дослідників: Л. Буєвої, Г. Бативощевої, С. Амеліної, Л. Барановської, О. Даниленко та ін. Когнітивні знання у галузі розробки програмних систем й формування професійних якостей інженерів з розроблення програмних систем проаналізували такі вчені: Б. Дурняк, Ф. Брукс, Г. Вейнберг, Н. Вірт, С. Гавенко, Т. Говорущенко, О. Кіриленко, Б. Ковальський, М. Коробчинський, М. Луцків, О. Лучшева, І. Огірко, В. Сеньківський, Л. Сікора, О. Тимченко, Н. Падалко, Г. Петріашвілі, О. Машков, Д. Щедролосьєв, та ін.

Розроблення комп'ютерних програмних систем у сучасних реаліях є масовою професією, але одночасно, на думку академіка А. Єршова, це одна з найважчих інженерних фахів. Складність її полягає у тому, що інженер-системотехнік має мати здатність першокласного математика до абстракції й логічного мислення у поєднанні з Едісонівським талантом споруджувати все що завгодно з нуля та одиниці. Складність розроблення та застосування у виробничому процесі програмних систем із використанням хмарної технології є основним чинником успіху у налагодженні бізнес-процесів із підтримки прийняття управлінських рішень.

Визначальну роль в успішності програмного бізнес-проекту є команда розробників, її склад, розташування і розподіл повноважень, а також нормативні обмеження та вимоги управління. Сьогодні не існує універсальної формули або технології для вирішення всіх проблем із розробки високонавантажувальних розподілених програмних систем. Такі системи на основі хмарної технології стають дедалі по-

пулярнішими. Багато процесів, які донедавна реалізовувалися на персональних комп'ютерах, зараз виконуються за допомогою Web-оглядача, прикладного програмного забезпечення та розподіленого доступу до різних онлайн-сервісів на різних незалежних обчислювальних платформах. Потреба у розподілених програмних системах на основі хмарної технології за останні десять років виросла настільки, що обчислювальних ресурсів одного сервера стає все більше не достатньо. Інтернет, а заодно і розподілені програмні системи проникли у всі галузі діяльності людини. Розподілені Web-сервіси чи програмні системи можна побачити у різних сферах діяльності людства, таких як енергетика, медицина, освіта, машинобудування і так далі. Поступово індустріальне суспільство відходить від класичного використання програмних систем, що здатні виконуватись тільки на обмежених архітектурних рішеннях та наявній мінімальній обчислювальній продуктивності.

Мобільність програмних систем – це основна потреба більшості користувачів сучасного інформаційного суспільства. Розподілені програмні системи дають можливість працювати на будь-яких сучасних інформаційних пристроях із будь-якого віддаленого місця, а отже, такі системи позбавлені класичного недоліку програмних систем.

На сучасному етапі інформатизації суспільства розроблення програмних систем із використанням хмарної технології виникає нагальна потреба в ефективному формуванні команд з врахуванням психотипів її членів для забезпечення мінімізації фінансових та часових ресурсів шляхом виявлення й усунення декількох центрів прийняття управлінських рішень, що створює *актуальну науково-прикладну проблему*. Використання методів групової динаміки має прямий вплив на згуртованість команди, що, своєю чергою, впливає на витрати з розробки програмного проєкту. Отже, необхідно розробити теоретичні та прикладні основ групової динаміки, що забезпечить можливість уникнення або мінімізації конфліктних ситуацій у процесі розроблення програмних систем із використанням хмарної технології.

Мета роботи – розроблення теоретичних та прикладних основ групової динаміки підвищення ефективності процесу програмних систем із використанням хмарної технології. Для досягнення поставленої мети необхідне створення інноваційної методології та інформаційної технології для правильного формування команд розробників програмних систем з огляду на їхні психотипи.

Для досягнення поставленої мети у роботі розв’язано такі завдання:

1. Проведено системний аналіз дослідження сучасного стану інформаційних технологій, аналіз впливу інформаційних потоків даних щодо якості розроблення програмних систем із використанням хмарної технології, дослідження ефективності використання моделей та архітектур.

2. Сформовано модель ієрархії процесу дослідження згідно з поставленим завданням.

3. Проведено системний аналіз наукових та літературних джерел згідно із завданнями дослідження.

4. Проведено дослідження життєвого циклу програмних технологій із метою визначення перспективних напрямів навчання та використання для розроблення програмних систем.

5. Деталізовано теоретичне обґрунтування використання інформації про програмні та базові технології «життєвого циклу» з метою розроблення програмних систем.

6. Проведено системний аналіз підходів із використанням методів групової динаміки до формування команд розробників програмних систем.

7. Розроблено та апробовано теоретичні й прикладні основи групової динаміки під час розроблення програмних систем із використанням хмарної технології.

8. Розроблено модель енеаграми визначення особистих рис кандидатів у члени команд розробників програмних систем та їх взаємозв’язків у робочій групі.

9. Досліджено вплив на групи розробників програмних систем у розрізі гендерного та національного різноманіття.

10. Удосконалено математичні моделі групової динаміки та взаємодії членів команд під час формування та функціонування групи розробників програмних систем.

11. Розроблено модель адаптивного тестування для визначення фахової компетенції претендентів у члени команд розробників програмних систем із використанням хмарної технології.

12. Проведено аналіз концепцій побудови високонавантажувальних розподілених відмовостійких Web-систем із використанням хмарної технології.

13. Удосконалено моделі та алгоритми архітектури розподіленої високонавантажувальної Web-системи із використанням хмарної технології як послуг для розроблення програмних систем.

14. Розроблено основи використання прикладного програмного забезпечення (проєктний менеджмент) як хмарного сервісу для управління розробкою програмних систем.

15. Розроблено методику визначення й подальшого врахування психотипів кандидатів під час формування команди розробників для забезпечення позитивного мікроклімату та ефективної роботи.

16. Проведено аналіз запропонованої гіпотези дослідження та практичної реалізації використання методів групової динаміки з огляду на психотипи членів під час формування ефективних команд розробників програмних систем.

Об'єктом дослідження є процес створення програмних систем із використанням методу групової динаміки з огляду на психотипи кожного члена команди для досягнення максимальної ефективності виконання поставлених завдань за мінімальних фінансових ресурсів.

Предмет дослідження – методи та засоби визначення психотипів претендентів у члени команди розробників програмних систем для формування стійких до збурень груп із мінімальним часом на адаптацію під час виникнення нештатних ситуацій.

Методи дослідження основані на використанні системного аналізу літературних та наукових джерел у розрізі актуальності науково-прикладної проблеми; адаптивних методах визначення компетенції претендентів у члени команд розробників; методах статистичного опрацювання життєвого циклу програмних технологій та сховищ даних із перспективою подальшого використання у розробленні програмних систем; методах групової динаміки для ефективного формування команд розробників програмних систем; методі енеаграм для визначення особистостей претендентів у члени команд розробників; методах динамічного програмування для визначення швидкості навчання та набуття фахових компетенцій членів команд розробників програмних систем із використанням хмарної технології; математичних методах динамічного функціонування програмних систем для ефективного використання у задачах широкого кола управління соціально-економічними системами; методах масштабування обчислювальних вузлів на незалежних обчислювальних платформах; методах організації кластерних обчислень для розроблення програмними систем; методах кешування інформаційних потоків даних для зменшення

обчислювального навантаження на програмну систему; математичних методах балансування обчислювального навантаження у розподілених програмних системах; методах лінійного програмування для порівняння та аналізу розроблених програмних систем; методах визначення оптимального зваженого співвідношення між використаними вихідними параметрами та вхідними факторами, які дають змогу визначати інтегральну оцінку параметрів критеріїв ефективності; методах кластерного та статистичного аналізу отриманих результатів для математичного опрацювання результатів дослідження; методах теорії прийняття управлінських рішень для формування ефективних команд.

Практичне значення отриманих результатів полягає у розробленні інформаційної технології створення команд розробників програмних систем із огляду на їхні психотипи, що забезпечує отримання комфортного мікроклімату та мінімізацію перманентних конфліктних ситуацій усередині команди. Застосування даної технології забезпечує підвищення якості розроблення програмних систем із використанням хмарної технології та значно скорочує час на адаптацію членів команд.

РОЗДІЛ 1

ДОСЛІДЖЕННЯ ТА ВИКОРИСТАННЯ ЗАСОБІВ ОПРАЦЮВАННЯ ІНФОРМАЦІЙНИХ ПОТОКІВ ДАНИХ У РОЗРОБЛЕННІ ПРОГРАМНИХ СИСТЕМ

1.1. Системний аналіз сучасного стану використання інформаційних технологій згідно з завданнями дослідження

Сучасний стан інформаційного суспільства пов'язаний із нагальною необхідністю використання інформаційних технологій, а саме збору, опрацювання й передачі великих обсягів інформації та ґрунтується на ній, що створило передумови переходу суспільства від індустріального до інформаційного. Глобальна інформатизація суспільства – це неперервний соціальний процес, в основі якого лежить провідна діяльність людства у сферах суспільного виробництва матеріальних благ, де продукування, збір, опрацювання, зберігання й передача інформаційних потоків даних забезпечується засобами сучасної комп′ютерної обчислювальної техніки. В інформаційному суспільстві акцентується комплекс інноваційних заходів, спрямованих на швидке отримання та застосування достовірних масивів даних, які є вичерпні та своєчасні для використання у всіх галузях діяльності.

Основною метою інформатизації сучасного суспільства є забезпечення процесів, що сприяють не тільки лавиноподібному науково-технічному прогресу, а й створенню інноваційного інформаційного середовища для зберігання та опрацювання значних потоків даних. Базовим критерієм інформатизації суспільства є створення програмо-технічних систем. Одним із головних напрямів розвитку інформаційного суспільства в Україні є індустрія, що дедалі зростає з розробки програмних систем на основі використання інноваційних конкурентоспроможних технологій. Оскільки застосування таких технологій значно розширює сферу використання багатьох потоків даних у різних галузях завдяки модернізації технологічних процесів та підходів до характеру праці, зокрема зменшення співвідношення фізичної

праці до розумової, мінімізації впливу людських помилок шляхом автоматизації бізнес-процесів.

Інформаційна технологія – це сукупність інформаційних процесів, що використовує моделі, методи, засоби та алгоритми накопичення, опрацювання і передачі інформації для отримання позитивного результату згідно запитами в інформаційному продукті [275].

Для ефективного застосування інформаційної технології необхідно заздалегідь забезпечити складний та тривалий процес підготовки початкових даних на складній обчислювальній комп'ютерній техніці. Таке опрацювання проводиться зі створенням ефективного програмного забезпечення завдяки застосуванню математичного апарату, в наслідок якого формуються відповідні інформаційні потоки даних для подальшого використання [264]. Одним з основних критеріїв для ефективного використання методології інформаційної технології є інтерактивність та динамічність у створенні структур і математичних функцій. В основі методології інформаційної технології лежать ґрунтовні дослідження з визначення базових структур даних та інформаційних процесів, які мають стійкі властивості й адаптивні характеристики інформаційної технології. Отже, такий підхід вимагає ґрунтовного аналізу можливостей як програмних, так і технічних засобів, а також технологічних процесів у заданій предметній галузі [309].

Сучасні вимоги до бізнес-процесів компаній полягають у дедалі більшому використанні різнорідних структурованих та неструктурованих інформаційних джерел на незалежних обчислювальних платформах, для забезпечення сталого економічного розвитку бізнесу та покращення якості надання послуг зацікавленим клієнтам. Ефективне управління інформаційними потоками даних стає критично важливою функцією для успішного ведення бізнес-процесів. Будь-яке матеріальне виробництво або надання інформаційних послуг потребує застосування програмно-технічних систем для опрацювання великих інформаційних потоків даних. В основі традиційних підходів та методів зазвичай лежать інформаційно-пошукові системи з використанням систем управління базами даних для підтримки прийняття управлінських рішень на базі бізнес-аналітики, яка формується із цих масивів.

Проте не потрібно недооцінювати проблему вибору інформаційної технології для розроблення програмних систем із використанням хмарної технології, а саме програмних засобів та комплексів управління базами і сховищами даних, оскільки від якості їхнього аналізу

та доцільності вибору буде залежати ефективність опрацювання великих інформаційних масивів даних. Із переходом в еру накопичення та опрацювання великих масивів інформації стає нагальна потреба у розробці ефективних методів аналізу, які будуть придатні для застосування через їх різнотипність, а також у фаховій підготовці значних обсягів багатьох спеціалістів задля підтримки бізнес-процесів. Водночас постійно зростає обчислювальна складність розроблених алгоритмів для підтримки прийняття управлінських рішень на основі лавиноподібного зростання обсягу зібраних та опрацьованих даних [274].

Водночас застосування відомих моделей, методів та засобів для аналізу великих інформаційних масивів даних не справджує очікувань розробників програмно-технічних систем, що, своєю чергою, створює значні перевитрати ресурсів й конфліктні ситуації через різницю між очікуваннями замовників та результатами, які вони отримали. Сьогодні загальносвітовий тренд у застосуванні інформаційних технологій – це опрацювання багатьох масивів даних, які дають змогу вирішувати сучасні задачі на базі наявних інноваційних технологій із застосуванням інтелектуального опрацювання інформації, а саме когнітивних методів, штучного інтелекту, семантичного аналізу та ін. [330].

Однак сьогодні не створено універсальних підходів до розробки і використання інформаційних технологій на базі програмно-технічних систем для будь-яких галузей діяльності людства, оскільки на змістовне наповнення та методи й алгоритми опрацювання інформаційних потоків даних суттєво впливає індивідуальна специфіка та особливості предметних галузей. На нашу думку, доцільно використовувати підхід, що ґрунтується на дослідженнях особливостей предметних галузей, а також використанні інноваційних підходів для розроблення програмних систем опрацювання великих інформаційних потоків даних із метою підтримки прийняття управлінських рішень у конкретних галузях. Водночас особливу увагу потрібно приділити напряму розроблення програмних систем із використанням хмарної технології, оскільки саме ефективне застосування програмних систем – це інформаційна база бізнес-діяльності сучасного суспільства [304].

1.2. Ієрархія процесу наукового дослідження

Для ефективного розроблення програмної системи першочерговим завданням дослідження є правильне формування програмних вимог та задач, які повинна вирішувати система. Розіб'ємо ці вимоги на етапи згідно з дослідженням із розроблення програмних систем з використанням хмарної технології, які позначимо R, а індекс при значенні буде визначати глибину етапу від 0 до 9 (рис. 1.1). Отже, нульовий етап R_0 буде відповідати за розробку програмної системи, а саме постановку задачі.

Деталізація ідеї розробки програмної системи та формування програмних вимог розглядаються на етапі R_1. На етапі R_2 розробники мають провести системний аналіз технологій, які необхідно буде використати під час проєктування системи, а саме програмних технологій, баз та банків сховищ даних, хмарних технологій. На цьому етапі буде здійснено дослідження у розрізі життєвого циклу технологій, перспективи їхнього застосування та прогнозу актуальності принаймні на п'ять подальших років. Наступний етап R_3, на нашу, думку не менш важливий, оскільки підбір претендентів у команди розробників можна проводити за допомогою традиційних підходів (тестування знань англійської мови, професійних знань) або засобами та методами групової динаміки, де, окрім традиційних підходів, ми як гіпотезу хочемо запропонувати визначення і використання психотипів претендентів для забезпечення більш комфортного клімату у колективі розробників. Проте сьогодні практично кожна розроблена програмна система використовує хмарні технології для мінімізації фінансових і зменшення людських ресурсів й від їхнього вибору буде залежати успішна реалізація проєкту. Ми проаналізували Java, C, C++, Python – мови програмування; Oracle, MySQL, MS SQL server, Postgres SQL – реляційні бази та банки даних; SaaS, PaaS, IaaS – хмарні технології. На етапі R_4 проводимо дослідження математичних моделей формування груп розробників програмних систем із використанням хмарної технології. Не менш важливим є дослідження та використання високонавантажувальних розподілених Web-систем для мінімізації мережевого навантаження завдяки різному кешуванню значних потоків даних, які відбуваються на етапі R_5. Також ми проаналізуємо автоматизоване розвантаження значних інформаційних потоків даних для швидкого та адекватного їх опрацювання. На етапі R_6 ми проведемо

дослідження щодо мінімізації фінансових та людських витрат на розроблення програмної системи із використанням хмарної технології. Етап R_7 забезпечить реалізацію програмної систем згідно з поставленими однотипними вхідними вимогами до розробки різними командами, сформованими як за традиційними підходами так із застосуванням методів групової динаміки. На етапі аналізу правильності програмної реалізації системи R_8 ми проведемо експериментальне дослідження нашої гіпотези про доцільність використання групової динаміки у процесі розроблення програмної системи.

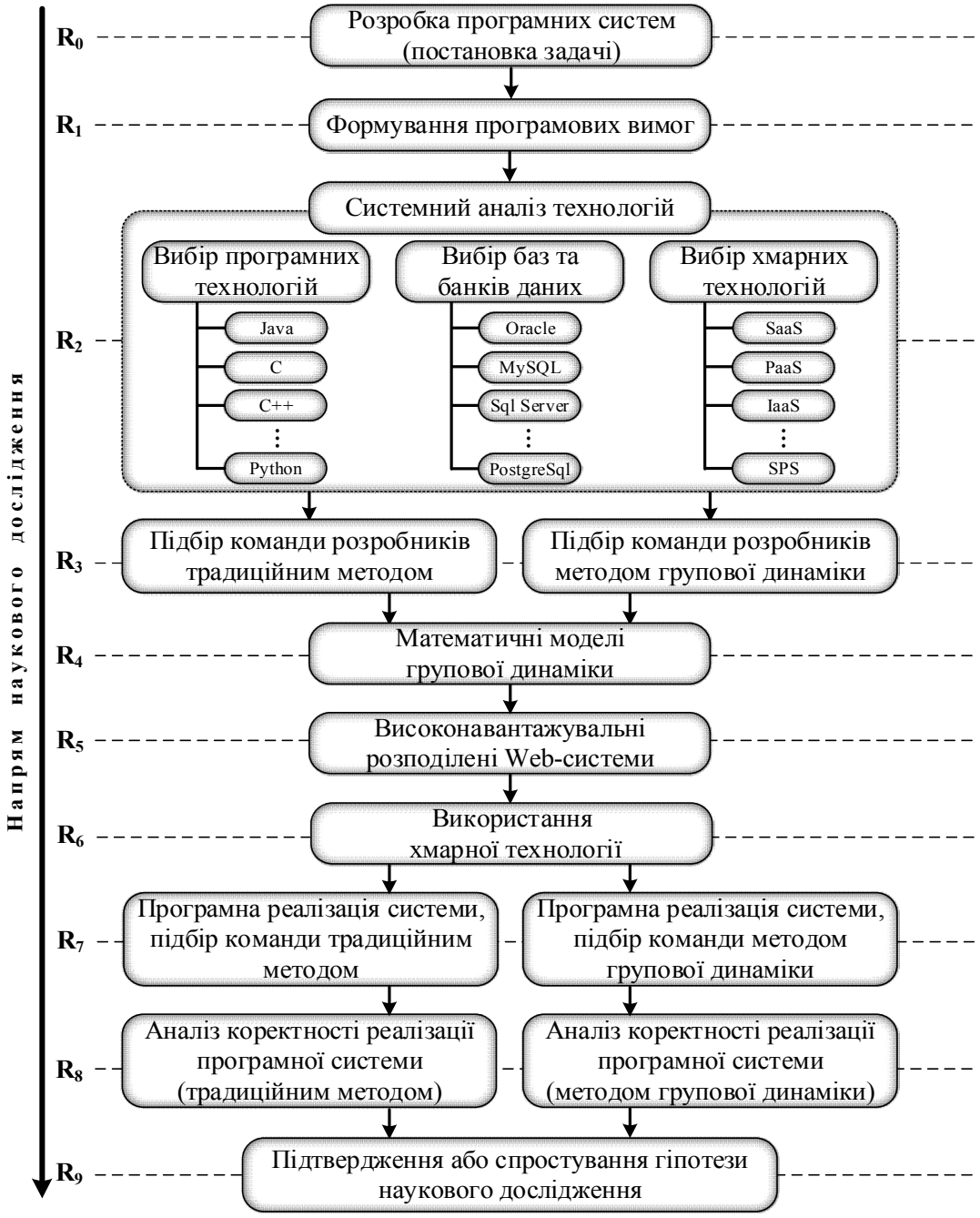


Рисунок 1.1 – Модель ієрархії процесу дослідження

Завершуючи наукове дослідження етапом R_9, де проведемо якісну оцінку по перевірці гіпотези доцільності використання методів групової динаміки, а саме визначення та врахування психотипу членів команд при їх формуванні для розроблення програмних систем з використанням хмарної технології.

1.3. Теоретичне обґрунтування життєвого циклу технологій

1.3.1. Програмні технології з розроблення прикладних систем

Кожна інформаційна технологія (ІТ) має свій життєвий цикл. У сучасному світі ІТ-технології розвиваються дуже динамічно і часто конкурують між собою або, навпаки, займають різні ніші, не вступаючи у відкриту боротьбу. Але очевидно одне, що розвиток сучасних інформаційних технології та рішень, побудованих за їхньою допомогою починають залежати від кількості кваліфікованих фахівці, залучених до розробки програмних систем із використанням хмарної технології, що безпосередньо їх застосовують та створюють комерційні рішення на її основі. Потрібно також враховувати промислове лобі, яке формує зацікавленість великих компаній у просуванні власних технологічних рішень із метою встановлення провідного становища у вже наявних та нових напрямах розроблення програмних систем із використанням хмарної технології.

Під терміном «технологія» потрібно розуміти, сукупність знань, відомостей про технологічний процес окремих виробничих операцій у процесі розроблення програмних систем або чогось іншого. Тому, як і в інших виробничих галузях, в ІТ-галузі базовою інформаційної технологію необхідно вважати не програмне забезпечення (ПЗ), а методи та засоби, за допомогою яких воно було створене, тобто мови програмування та технологічне середовище, яке утворилося довкола розробленого комерційного програмного продукту.

Водночас кожна мова програмування (технологія) чи системи керування базами даних (сховищами даних) розробляється під конкретну бізнес-логіку, тому безпосереднє порівняння різних за ідеологією мов є не доцільним, відтак було б варто враховувати обґрунтування

показників їхнього життєвого циклу (ЖЦ) до початку розроблення програмних систем з використанням хмарної технології.

На сучасному ринку існує багато інноваційних інформаційних технологій та технологічних рішень для розроблення програмних систем із використанням хмарної технології, проте немає узагальненої класифікації для оцінки їхньої перспективності, можливість впровадження, доцільності застосування й кількості працівників, які мають або матимуть у майбутньому достатній рівень кваліфікації для створення та промислової підтримки таких систем. Ключовою проблемою, яка постає перед командою з розробників програмних систем із використанням хмарної технології, є компетентність менеджерів, керівників фірм, компанії, великих корпорацій визначити, яку технологію необхідно використати із запропонованих (доступних на ринку), щоб забезпечити достатньо якісну продуктивність розроблення комп'ютерних систем та мінімізувати фінансові і часові витрати, аби в майбутньому отримати необхідну кількість кваліфікованих працівників, які б могли підтримувати програмний продукт, а водночас і вдосконалювати його з найменшими витратами, які невпинно зростають. Цей ріст пов'язаний із стрімким розвитком інноваційних інформаційних технологій та інформатизації суспільства й проникненням програмних систем у повсякденне життя користувачів.

Зараз життєвий цикл технології розглядається у контексті програмного продукту, тобто не існує певного розділення між технологією і програмним продуктом. Вважається, що інформаційна технологія – це теж програмний продукт, який більше використовується через ефект «лавину» – кількість сучасних програмних продуктів, які були розроблені та які потрібно підтримувати, не дають змоги відмовитися від навіть дуже застарілої технології. Наприклад, мови програмування C/C++ є достатньо успішними навіть після 30–40 років свого існування. Їхня частка на ринку зменшилася не через те, що вони втратили актуальність, а тому, що інші напрями почали більше застосовувати нові інформаційні технології. Вони стали вузькоспеціалізованими технологіями і водночас перейшли на останній етап життєвого циклу свого існування, значно уповільнивши свій розвиток та водночас зменшивши кількість кваліфікованих спеціалісті із таких технологій. Неприпустимо вважати, що найпопулярніша зараз технологія буде популярною через 10–15 років. Такий випадок можливий, проте величина ризику такого твердження є колосальною. До 2000 року C++ вважався еталоном для розробки більшості програмного за-

безпечення або програмних систем, а кількість програмістів, які його опанували на високому рівні, була великою тоді інформаційна технологія Java була на початковому етапі свого життєвого циклу з малою кількістю відповідних кваліфікованих спеціалістів, що могли взаємодіяти з нею, складно було уявити, що за 10 років вона перевершить за кількістю спеціалістів C++ у декілька разів. Причиною цього була висока швидкість розробки програмних систем із використанням хмарної технології та зручною адаптивністю під різні незалежні обчислювальні платформи, легкість освоєння порівняно з C та C++.

Розроблення програмних систем із використанням хмарної технології мають високу цінність тільки тоді, якщо їх створюють із перспективою (3–5 років), тобто на початковому етапі використовуються матеріальні та людські ресурси з метою отримання максимального доходу в майбутньому у продовж тривалого часу. Вибір інформаційної технології, за допомогою якої будуть розроблятися програмні системи, вкрай важливий. Оскільки не існує певних охарактеризованих параметрів, на які потрібно звернути увагу, а тільки розмиті уявлення у різних представників ринку із розроблення програмних систем, то доцільно теоретично обґрунтувати та виділити основні параметри оцінки життєвого циклу перспективної інформаційної технології, щоб не потрапити у ситуацію переходу від однієї технології до іншої під час розроблення або експлуатації цієї системи.

Отже, під час вибору інформації технології потрібно оцінити, на якому етапі життєвого циклу свого існування вона перебуває у той чи інший час і чи для програмної системи, яку розробляють, цей етап є доцільним для подальшого використання. Для оцінки етапу життєвого циклу інформаційної технології було проведено системний аналіз параметрів, за якими можна охарактеризувати перспективні технології для використання, та їх якісних і кількісних характеристик. Усі параметри були поділені на дві групи: параметри, які безпосередньо та опосередковано характеризують етапи життєвого циклу інформаційної технології. На основі такої класифікації було створено узагальнену систему перспективності використання цих технологій.

Наявні критерії життєвого циклу оцінки перспективності інформаційних технологій для розроблення програмних систем із використанням хмарної технології не відображають етапи розробки, а також стан та доцільність тієї чи іншої технології у певний проміжок часу. Тому було виділено такі параметри, які характеризують етап життєвого циклу технології:

- період існування на ринку;
- парадигма;
- кількість кваліфікованих спеціалістів із цієї технології;
- популярність засобів розробки на цій технології;
- швидкість адаптації до сучасних умов;
- величина спільноти розробників програмних систем.

Своєю чергою, було також виділено опосередковані параметри:

- швидкість опанування тієї чи іншої технології;
- підтримка перспективних інформаційних технологій великими компаніями.

Наявний перелік критеріїв оцінки технологій не можна вважати кінцевим або остаточним під час вибору перспективної технології, але перераховані показники розкривають «підводне каміння» під час її життєвого циклу. Провівши системний аналіз життєвого циклу різних інформаційних технологій, а саме періоду існування програмних технологій для розроблення програмних систем із використанням хмарної технології, можна із певною ймовірністю стверджувати про подальшу перспективу її застосування на ринку IT-галузі. Ці параметри безпосередньо характеризують етап життєвого циклу різноманітних інформаційних технологій, тому можна стверджувати, що технології віком 5–10 років можна вважати молодими, 10–20 років – зрілими, 20–30 років – такими, що стрімко старіють (рис. 1.2).

Рисунок 1.2 – Тенденція популярності мов програмування, основана на оцінюванні кількості запитів у мережі Інтернет (2005–2018)

Під час наукового дослідження провели системний аналіз найбільш популярних мов програмування (програмних технологій), що висвітлюють різні напрями розроблення програмних систем із використанням хмарної технології: Java і C# – інтерпретуються у проміжний код віртуальної машини, Ruby та Perl – безпосередньо інтерпретуються під час виконання, C+ – компілюється до виконання на обчислювальних платформах. Встановили, що програмна технологія Java із запуску релізу першої стабільної версії в 1995 році стрімко зростає, тобто належала до молодого покоління інноваційних програмних технологій, а з 2005 року перейшла у стадію зрілості згідно з життєвим циклом. Програмна технологія C# вийшла на ринок IT-галузі у 2001 році, а в період 2001–2012 рр. демонструвала стрімке зростання кількості фахових спеціалістів, які опанували її (молоде покоління). Проте програмна технологія C++ отримала першу стабільну версію у лютому 1985 року (перший зовнішній випуск) та демонструвала стрімке зростання кількості фахових спеціалістів майже до 1998 року.

Перша загальнодоступна версія програмної технології Ruby вийшла на ринок IT-галузі у 1995 році, впродовж 1995–2007 рр. перебувала на стадії «молодості» існування програмної технології. У стадію зрілості вона перейшла приблизно в 2007 році. Програмна технологія Perl є однією із перших серед мов інтерпретованих для розроблення програмних систем, оскільки перший стабільний реліз вона отримала у 1987 році. У її життєвому циклі молодість випала на початок ери створення мережі Internet, а зрілості–на період 1994–2005 рр., проте з 2005 року вона втратила популярність та перестала цікавити розробників.

Підсумовуючи наше системне дослідження, зазначимо що цей параметр перебування програмних технологій на тому чи іншому етапі життєвого циклу є особливо важливим критерієм у контексті перспективи застосування їхнього розроблення програмних систем із використанням хмарної технології. Наприклад, якщо обрати програмну технологію, опираючись тільки на час присутності її на ринку IT-галузі, то присутність більше ніж 10 років – це очевидний вибір. Але для продуктів, які розробляються на перспективу, – це хибний шлях, оскільки кількість кваліфікованих спеціалістів із цієї програмної технології невпинно починає знижуватися через високу конкуренцію серед інших програмних технологій, що будуть більш привабливими завдяки своїй новизні та мобільності. Тому для розроблення короткострокових програмних систем (проєктів) зрілі програмні технології –

це найкращий варіант вибору, а для розроблення довгострокових програмних систем потрібно обирати технології, що належать до молодої категорії життєвого циклу, але водночас є перспективними з позиції інших параметрів, які розглядаються.

Парадигма розроблення програмних систем із використанням хмарної технології виступає способом концептуалізації, що визначає організацію програмних обчислень та структурування роботи членів команд розробників (програмних систем), яку виконує програмно-технічний комплекс.

Внаслідок проведення системного аналізу життєвого циклу програмних технологій було встановлено, що ця парадигма, наведено на (рис. 1.3), характеризує інноваційні підходи у розробленні програмних систем із використанням хмарної технології, проте не є одним із ключових факторів, оскільки вона супроводжує її у продовж усього часу існування, проте за її допомогою можна безпосередньо визначати період її існування.

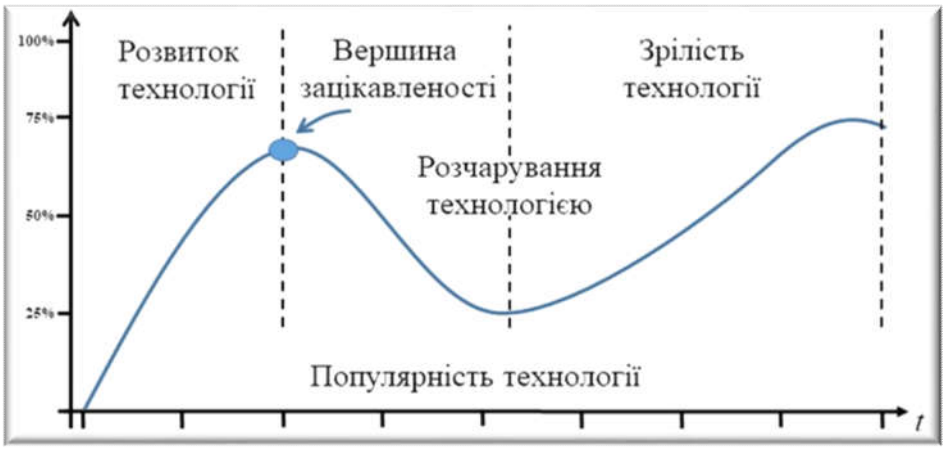


Рисунок 1.3 – Типовий цикл інформаційної технології та її етапи

Тому розроблення мов програмування (програмних технологій) до 1990 року проводилася лише за підтримки однієї або іншої парадигми, імперативної чи об’єктно-орієнтованої, а після 1990 року отримали, ще імперативну, об’єктно-орієнтовану, функціональну, декларативну та узагальнену. Хоча більшість мов програмних технологій декларують підтримку різноманітних підходів, все-таки розроблення програмних систем із використанням хмарної технології здійснюється на основі того, яка саме із них визначена розробниками як основна.

Сьогодні найбільш популярні мови для розроблення програмних систем (рис. 1.4) належать до об’єктно-орієнтованої парадигми.

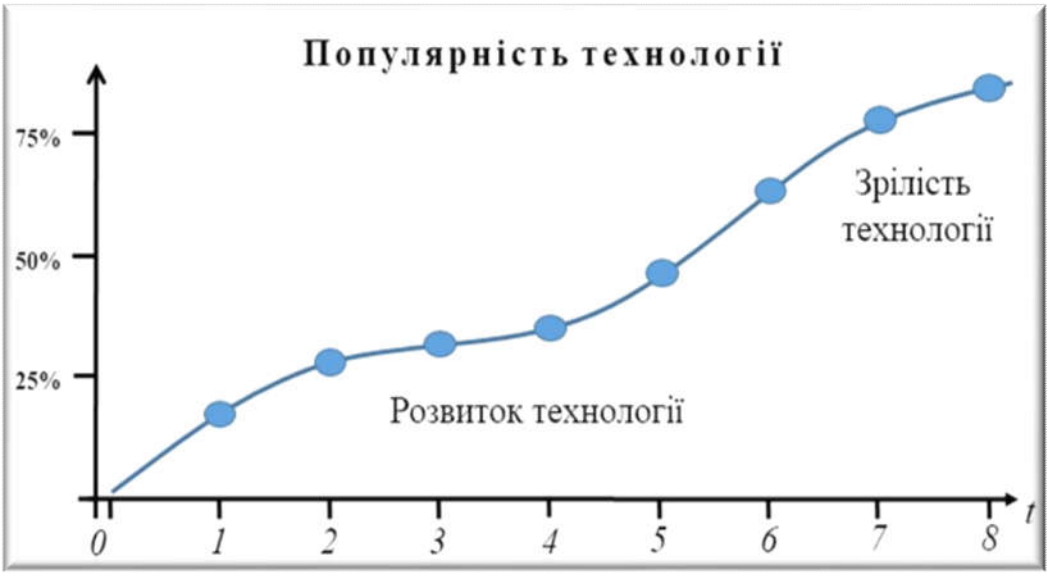


Рисунок 1.4 – Стандартний цикл інформаційної технології

Використання цієї парадигми безпосередньо не вказує командам розробників програмних систем, на якому саме етапі життєвого циклу програмної технології вона перебуває зараз, проте вказує на тривалість її циклу. Проте об’єктно-орієнтовані програмні технології мають суттєвий недолік, який поряд із забезпеченням їм значного періоду існування передбачає створення великої кількості залежностей, що створюються у виробничому процесі програмної розробки комп’ютерних систем. Відповідно до вище зазначеного, доречним є вислів Джо Армстронга в книзі «Coders at Work»: «Проблема об'єктно-орієнтованих мов у тому, що вони тягнуть з собою все своє неявне оточення. Вам потрібен був банан – а ви отримуєте горилу з бананом, і цілі джунглі на додачу». Тобто у процесі розроблення програмних систем з використанням хмарної технології застосовуючи об’єктно-орієнтовані парадигми програмування команди розробників значно розширюють час підтримки цієї технології, а значить присутність її на ринку значно зростає. Виникає парадоксальний висновок: «Парадигми, що тягнуть за собою своє оточення, володіють довшим життєвим циклом».

Життєвий цикл програмних технологій якісно залежить від кількості кваліфікованих спеціалістів, які опанували ці програмні технології. Такий параметр, на нашу думку є визначальним та окреслює, як зацікавленість і компетентність команд розробників із певної програмної технології визначає рівень конкуренції на ІТ-ринку. Велика пот-

реба у фахових спеціалістах, що опанували певну програмну технологію, спричиняє появу різноманітних автоматизованих засобів розроблення комп'ютерних програмних систем. Проте, ґрунтуючись на дослідженнях життєвого циклу програмних технологій, засвідчує, що цей зв'язок не односторонній, оскільки зацікавлень у спеціалістах певної програмної технології входить у пряму залежність від наявних автоматизованих засобів розробки, властивостей та кількості фахових спеціалістів ІТ-галузі.

Це замкнуте коло спричиняє ефект «вічної молодості» певної програмної технології, а саме не здатність її отримати критичну масу автоматизованих інструментів та відповідних спеціалісті, що у подальшій перспективі забезпечать їй популярність та перехід у сектор «вічно перспективних» програмних технологій. Також слід відмітити, що ефект «вічної молодості» не відміняє такий важливий параметр як, «період існування даної програмної технології на ринку», тобто після 20–30 років концепції, які були закладанні при проектуванні програмних технології стають не достатньо актуальними або неінноваційними для розроблення програмних систем з використанням хмарної технології. Зважаючи, що основна маса фахових спеціалістів зосереджена у межах 10–16 мов програмування (програмних технологій), то їх список є динамічний та постійно змінюється у процесі еволюції ІТ-ринку наведено на (рис. 1.5).

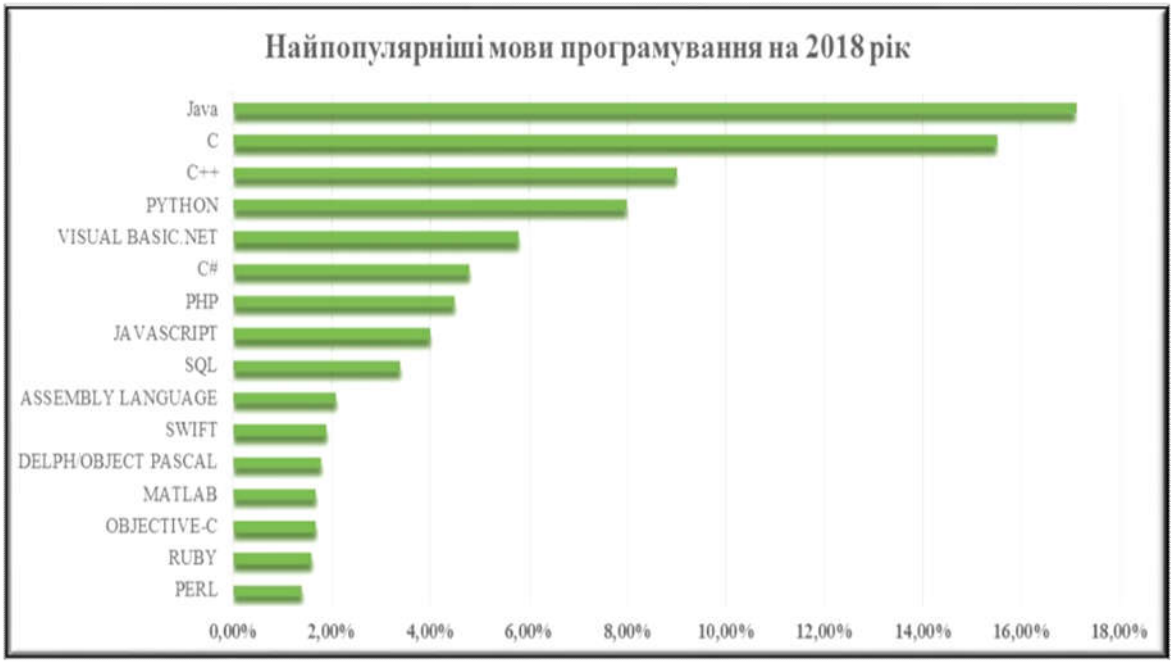


Рисунок 1.5 – Найпопулярніші програмні технології станом на 2018 р.

Отже, для визначення критерію перспективності та етапу життєвого циклу тієї чи іншої програмної технології для розроблення високонавантажувальних розподілених систем потрібно враховувати такий показник, як загальна кількість спеціалістів із цієї технології і відстежувати тенденцію росту їхньої кількості на IT-ринку. Якщо їх багато, то приріст числа незначний або його немає, тобто програмна технологія перебуває на піку своєї популярності. Водночас мала кількість спеціалістів разом із колосальними темпами зростання їх висвітлюють перехід технології зі стадії молодості на стадію зрілості, що є оптимальним варіантом для розроблення програмних систем із використанням хмарної технології, які розробляються на перспективу.

Популярність засобів розроблення високонавантажувальних розподілених систем на цій технології не враховують під час оцінки, оскільки вважається, що популярність засобів розроблення програмних систем прямо залежить від кількості фахових спеціалістів та часу існування на IT-ринку. В історії розвитку IT-ринку програмних технологій часто траплялися ситуації «другого дихання» мов програмування, тобто коли популярність зростала не завдяки інноваційним особливостям самої програмної технології, а завдяки появі інноваційних автоматизованих засобів розробки у цій мові програмування. Наприклад, програмна технологія Ruby до 2004 року демонструвала помірковане зростання на IT-ринку, але в період з 2004 до 2006 року, згідно з індексом TIOBE (TIOBE programming community index), збільшила свою присутність на ринку втричі та здобула перемогу в номінації «Мова програмування року» [178].

Аналогічна ситуація повторилась і для програмної технології Python з її фреймворком Django, після появи якого відбулося суттєве зростання популярності й активізувалися зусилля до розробки альтернативних засобів програмних систем, що водночас значно розширило область застосування та збільшило привабливість для розроблення програмних високонавантажувальних розподілених систем. Також потрібно враховувати, що поява інноваційних (автоматизованих) засобів розробки позитивно впливає на ще один фактор, а саме на час присутності на IT-ринку цієї мови програмування.

Отже, можна впевнено стверджувати, що значна популярність певної програмної технології та її великий і альтернативний набір інструментальних засобів розробки відтермінує етап стрімкого старіння, що важливо під час оцінки життєвого циклу мови програмування, як технології, так і її вибору для розроблення програмних систем із

використанням хмарної технології. Консолідовані дані популярності тієї чи іншої програмної технології наведено у табл. 1.1, у ній відображено переможців у номінації «Мова програмування року» згідно з індексом TIOBE за період з 2003 до 2018 року, а також рік появи знакових програмних технологій для цих мов програмування як засобів розроблення.

Таблиця 1.1 – Мова програмування року згідно з індексом TIOBE та засоби програмної розробки, що сприяли зростанню популярності

Рік	Мова програмування року	Засіб програмної розробки	Рік розробки
2018	Java	Spring Hadoop	2012
		Spring Boot	2014
2014	JavaScript	AngularJS	2009
		Backbone	2010
		Meteor	2012
		ReactJS	2013
2013	Transact-SQL	-	-
2012	Objective-C	Apportable SDK	
2011	Objective-C	iOS SDK	2009
2010	Python	Python 3	2009
2009	Go	-	-
2008	C	-	-
2007	Python	Django	2003
2006	Ruby	Ruby on Rails	2004
2005	Java	Spring Framework	2003
2004	PHP	WordPress	2003
2003	C++	Boost (libraries)	2001

Одним із важливих аспектів програмної технології є швидкість її адаптації до промислових бізнес-вимог. В умовах інформаційного ринку, що динамічно розвивається, здатність програмної технології гнучко пристосовуватися до потреб IT-ринку досить недооцінена, але з огляду на те, що швидка адаптація може ламати зворотну сумісність процесу еволюції, що відбувається достатньо повільно. Також потрібно враховувати такий критерій, як швидкість адаптації програмної технології до потреб зацікавлених осіб (замовників розроблення про-

грамних систем), які виникають у процесі розроблення високонавантажувальних розподілених програмних систем, що накладають на розробників суттєві технологічні обмеження.

Для всіх програмних технологій швидкість еволюції та напряму їх розвитку буде залежати від потреб інформаційного ринку та ніші, у якій вони існують, але загалом виділити два підходи розвитку:

- повна зворотна сумісність;
- зворотна сумісність у межах наявної версії.

До переваг першого підходу можна зарахувати такі критерії:

- програмне забезпечення, написане за допомогою старішої версії, буде працювати на новішій;
- непотрібно оновлювати програмне забезпечення відповідно до нової версії програмної технології;
- менша потреба у вивченні спеціалістами нових особливостей та підходів до розроблення програмних систем із використанням хмарної технології.

До недоліків першого підходу належать такі критерії:

- зменшення швидкості розроблення програмних продуктів порівняно з аналогічними, що розробляються на новіших програмних технологіях;
- менша кількість «синтаксичного цукру» (англ. Syntactic sugar) порівняно з іншими технологіями;
- постійна міграція багатьох програмістів на інші технології;
- низька швидкість адаптації до потреб інформаційного ринку.

Серед переваг другого підходу відзначимо такі критерії:

- кодова база програмного забезпечення відповідає найновішим практикам розроблення програмних систем із використанням хмарної технології;
- перехід багатьох програмістів з інших технологій;
- велика кількість «синтаксичного цукру», що значно прискорює процес розробки;
- збільшення життєвого циклу програмної технології;
- висока конкурентоспроможність цієї програмної технології на IT-ринку.

Другий підхід характеризується такими недоліками:

- програмна технологія працює лише у межах тієї програмної версії, за допомогою якої вона розроблялася;
- необхідність оновлювати код програмного забезпечення під час оновлення до новішої програмної версії мови програмування, яка,

своєю чергою, порушує зворотну сумісність;

- необхідність вивчення розробниками програмних систем нових підходів із розробки для забезпечення потреб бізнес-логіки замовника та витрачати додатковий час на це вивчення.

З огляду на переваги та недоліки використання певних технологічних рішень, а саме, що повна зворотна сумісність є безумовною перевагою на стадії зрілості, але у той же час в загальному зменшує швидкість розробки та сповільнює розвиток програмної індустрії у цілому. Отже, критерій сумісності у межах певної версії є цілковитою протилежністю першого підходу, оскільки забезпечує значне підвищення швидкості розроблення програмних систем із використанням хмарної технології завдяки застосуванню інноваційних підходів, навіть якщо вони порушують зворотну сумісність. Сьогодні дедалі частіше використовують другий підхід, але з деякими технологічними обмеженнями, тобто адаптацію програмного коду до нової версії програмної технології проводять лише у тому випадку, коли це забезпечать кардинальне покращення продуктивності та швидкості розроблення програмної системи з використанням хмарної технології.

Отже, результат розроблення програмної системи у межах однієї версії забезпечує переваги першого підходу, а перехід на нові версії забезпечує переваги другого підходу. Критерій швидкості адаптації безпосередньо не характеризує життєвий цикл, але вказує, на скільки довго буде існувати ця програмна технологія.

Визначення критерію величини програмної спільноти, можна оцінити за декількома властивостями, які має програмна технологія:

- популярність програмної технології на цьому етапі;
- перебування програмної на етапі життєвого циклу;
- перспективи розвитку та використання програмної технології.

Щоб оцінити стан етапу життєвого циклу програмної технології, доцільно розглянути динаміку росту певної програмної спільноти ІТ-ринку. Якщо величина спільноти почала рости дуже повільно або взагалі скорочується, то технологія перейшла свій пік і перебуває на середині періоду зрілості. Вона ще буде популярна у межах зайнятої ніші доти, поки кількісне відображення обсягу активної програмної спільноти ІТ-ринку не почне стрімко скорочуватися. Такий процес є вихідною точкою етапу старіння певної програмної технології. Також ця характеристика відображає зацікавленість фахових спеціалістів у такій програмній технології та її перспективах використання на інформаційному ринку для задоволення потреб бізнес-логіки замовників.

Щоб оцінити перспективи використання певної програмної технології, достатньо порівняти динаміку росту обраної та тих, що вже завоювали прихильність під час етапу молодості. Якщо динаміка схожа, то і перспективи теж схожі, оскільки однакові темпи зростання перспективної технології, що вже завоювала IT-ринок, здатна свідчити про можливість циклічного повторення успіху.

Під час дослідження було виділено такі параметри, які не характеризують програмні технології з погляду етапу життєвого циклу так і доцільності обрання технології на різних етапах існування, але безпосередньо впливають на тенденції розвитку розроблення програмних систем із використанням хмарної технології. До таких параметрів можемо зарахувати швидкість та легкість освоєння певної програмної технології, а також всебічну підтримку великими компаніями.

Швидкість освоєння певної програмної технології є дуже критичним параметром, оскільки потреба у фахових спеціалістах із розроблення програмних систем із використанням інноваційних технологій зростає значно швидше, ніж освітні установи можуть у достатній мірі адаптуватися до підготовки відповідних кваліфікованих спеціалістів. Тому легкість освоєння програмної технології у багатьох випадках стає тим фактором, який забезпечує ріст програмної спільноти на IT-ринку, а також появу інноваційних засобів розробки та водночас тривалий період існування цих програмних технологій. Складність розроблення програмних систем із використанням хмарної технології починає зростати значно швидше, ніж продуктивність програмістів.

Велика кількість абстракцій у програмних технологіях, які були популярні під час розробки програмних додатків за останні 30 років, призвели до того, що багато патернів проєктування приховані, а отже, незрозумілі. Це породжує складність у розумінні молодими спеціалістами того, як той чи інший програмний механізм працює. В умовах дефіциту спеціалістів на цей критерій слід зважати на стан оцінки програмної технології при виборі її. Перехід багатьох спеціалістів із розробки програмного забезпечення за допомогою C++ на Java-технологію відображає цю проблему. При абсолютній перевазі C++ у плані продуктивності, вимогам до пам’яті, величині програмної спільноти та кількості засобів автоматизованої розробки Java володіє трьома превагами такими, як швидкість розроблення програмних систем, легкість освоєння даної програмної технології.

Всебічна підтримка великими компаніями певної програмної технології полягає у стрімкому зростанні на початкових стадіях її вико-

ристання, які зосереджені на внутрішніх продуктах та на використанні власної програмної технології, що в цілому її розвиває. Також суттєвий вплив на життєвий цикл програмної технології відіграють фінанси, які вкладаються у її популяризацію, розвиток та навчання спеціалістів. Великі компанії підтримують розроблену програмну технологію та зацікавлені у тривалому терміні її існування на IT-ринку. Для таких програмних технологій, як Java, C#, Swift, Objective-C, підтримка батьківських компаній була суттєвою в їх розвитку та позиції на IT-ринку. Тому при схожості вихідних параметрів різних програмних технології підтримка великих компаній є суттєвою превагою для розвитку та просування.

1.3.2. Використання баз та банків даних для розроблення програмних систем

На сучасному етапі інформатизації суспільства жодна розробка програмної системи не обходиться без використання баз та банків (сховищ) даних, які є важливим компонентом в інженерії програмних систем із використанням хмарної технології для сучасного суспільства. Більшість із нас зустрічаються з декількома базами або сховищами даних щодня, що пов'язано з деякою взаємодією програмних систем з інформаційними масивами даних. Наприклад, якщо ми йдемо до банку для оформлення депозиту або знімати кошти, якщо замовляємо номер у готелі або бронюємо квитки в авіакомпанії, якщо ми маємо доступ до комп'ютеризованого бібліотечного каталогу для пошуку бібліографічних елементів або якщо ми купуємо щось онлайн (книги, іграшки чи комп'ютери). У такому випадку виникає велика ймовірність того, що наша діяльність залучить якусь базу або сховище даних для вирішення нашого нагального питання. Навіть придбання товарів у супермаркеті часто автоматично фізично оновлює базу даних, яка містить інвентаризацію продуктових товарів.

Ці взаємодії є прикладами того, що ми можемо назвати традиційною базою або банком даних підчас розроблення програмних систем, у яких більша частина інформації, що зберігається і є доступною, текстова або числова. Протягом останніх декількох десятиліть досягнення в IT-галузі різноманітні техніки з розроблення програмних систем призвели до створення інноваційних систем управління базами даних. Стрімкий розвиток (популярність) серед мільйонів користувачів соціальних сервісів вимагав від соціальних медіа, таких як Facebook та

Twitter, серед багатьох інших програмних вимог і постало завдання по створенню величезних баз та банків даних, що зберігають структуровані та неструктуровані (нетрадиційні) дані, зокрема повідомлення, твіти та відеокліпи. Нові типи систем управління базами та банками даних часто називають великими даними (BigData) системи зберігання даних або системи NoSQL (документоорієнтовані бази даних), створені для управління даними соціальних мультимедійних програмних систем із використанням хмарної технології. Ці типи програмних систем, які застосовують системи управління базами та банками даних, також використовують такі компанії, як Google, Amazon і Yahoo, щоб керувати значними потоками інформаційних даних, які необхідні для Web-пошуку і для функціонування хмарних сховищ, за допомогою яких зацікавлені користувачі отримують колосальні можливості у мережі Інтернет для керування та опрацювання усіма різнорідними типами даних, включаючи документи, програми, зображення, відео та електронні листи.

Розглянемо нові можливості застосування систем управління базами та банками даних на незалежних обчислювальних платформах, використавши хмарні технології. Вдаючись до системи зберігання даних та онлайн-аналітичного опрацювання (OLAP), багато компаній отримують і аналізують корисну бізнес-інформацію з великих інформаційних масивів потоків даних для підтримки прийняття управлінських рішень та забезпечення вимог бізнес-логіки. Ці інформаційні технології застосовуються у реальному часі, а активна база або банки даних використовується для контролю промислових та виробничих процесів. Також інноваційні методи пошуку запитів користувачів корисні у базах для правильного знаходження корисної інформації у Всесвітньому павутинні, які переглядають бази та банки даних у мережі Інтернет.

Використання інформаційних технології, зокрема баз та банків даних, забезпечило значний вплив на стрімкий розвиток комп’ютерних програмно-технічних комплексів. У результаті системного аналізу можна стверджувати, що бази та банки даних відіграють вкрай важливу роль практично у всіх сферах діяльності, де використовуються комп'ютери, разом із бізнес-логікою, електронною комерцією, соціальними медіа, медициною, генетикою, правом, освітою та бібліотечною наукою.

Однак бази та банки даних, окрім очевидної перевірки мають ще такі властивості:

1. Бази та банки даних – це деякий аспект реального світу, який іноді називається мінісвітом або всесвітом дискурсу (UoD), тобто зміни в мінісвіті відображаються у базах або банках даних.

2. Бази та банки даних – це логічно зв'язаний набір інформаційних даних із деякими невід'ємним сенсом. Випадковий набір даних не може бути належно опрацьованим та названим базою даних.

3. Бази та банки даних розроблені, побудовані та заповнені інформаційними потоками даних для поставлених конкретних цілей.

Прикладом використання під час розробки програмної системи великої комерційної бази даних є електронний ресурс Amazon.com. Він містить інформаційні потоки даних для більше ніж 60 мільйонів активних користувачів, мільйони книг, компакт та DVD-дисків, ігор, електроніки, одягу й інших предметів. База та банк даних займає понад 42 терабайт (один терабайт містить 1012 мегабайтів) і зберігається на сотнях комп'ютерів на незалежних обчислювальних платформах, які називаються серверами або кластерними серверами. Мільйони користувачів мають змогу відвідувати Amazon.com будь-коли, використовуючи водночас бази та банки даних для задоволення своїх потреб. Ця база забезпечує постійне оновлення актуальної інформації, зокрема надходження нових книг і інших елементів, які додаються до неї, а запаси запасів оновлюються під час здійснення транзакцій.

Схема бази або банку даних для розроблення програмних систем із використанням хмарної технології може генеруватися, підтримуватися вручну, так і за допомогою автоматизованих системи управління. Наприклад, каталог бібліотечних карт є базою даних, яку можна створити та опрацювати вручну, а також можна створювати й підтримувати комп'ютеризовану базу даних групою прикладних програм, які написані спеціально для виконання цього завдання, або ж системою управління базами даних. Звичайно, ми взаємодіємо лише з інформаційними технологіями під час розробки програмних систем у нашому дослідженні.

Спільне використання та доступ до бази даних дає змогу багатьом користувачам і програмам отримувати миттєвий канал зв’язку до інформаційних потоків даних одночасно. Прикладна програмна система звертається до бази даних шляхом надсилання користувацьких запитів або запитів даних до систем управління базами даних, а запит зазвичай формує та викликає необхідні дані. Такий процес опрацювання даних називається транзакцією. Інші важливі функції, що надаються системою управління базами даних, охоплюють вери-

фікацію та захист бази даних, а також підтримання її у працездатному стані (цілісність бази даних) протягом тривалого періоду. Отже, система захисту запобігає виникненню дефектів (або аварій) в апаратному чи прикладному програмному забезпеченні та захисті від несанкціонованого або шкідливого доступу до інформаційних потоків даних. Типова велика база даних може мати досить великий цикл продовж багатьох років, тому система управління базами даних повинна мати змогу здійснювати масштабування системи в цілому, дозволяючи їй розвиватися у міру зміни бізнес-вимог протягом тривалого часу.

Для програмної реалізації систем не є абсолютно необхідним використання систем управління базами даних загального призначення в інформаційних системах зберігання та опрацювання інформаційних потоків даних. Можна написати спеціальний набір програм (сервісів) для створення та підтримки бази даних, фактично створюючи спеціальне програмне забезпечення систем управління базами даних для конкретних додатків, наприклад, таких як бронювання квитків в авіакомпанії. У будь-якому випадку, коли ми використовуємо систему управління базами даних загального призначення і незалежно від цього, нам необхідно розгорнути чимало прикладного програмного забезпечення для забезпечення адекватної роботи розробленої програмної системи з використанням хмарної технології. Насправді більшість систем управління базами даних є дуже складними програмними системами.

Для більш детального розуміння, яку ж саме систему управління базами даних нам необхідно вибрати для розроблення програмної системи, обчислимо значення популярності системи шляхом стандартизації й усереднення індивідуальних параметрів. Ці математичні перетворення зроблені так, що зберігається відстань між окремими модулями програмної системи. Це означає, що коли система А має в двічі більшу величину в DB-системах, ніж позиціонування у системі В, то вона у двічі популярніша за усереднений показник між окремими критеріями оцінки. Для усунення ефектів, спричинених зміною кількості джерел даних, ми погоджуємося на той факт, що оцінка популярності певної технології завжди має відносне значення, яке має бути пояснено порівняно тільки з іншими системами.

DB-Engines рейтинг вимірює кількість встановлених систем або їх використання у межах IT-галузі. Можна очікувати, що підвищення популярності системи зберігання та опрацювання даних, які ми вимірюємо за допомогою DB-Engines ранжирування, у дослідженні пере-

дують відповідному широкому визначенню критерію застосування системи за певний інтервал. Тому DB-Engines рейтинг може виступати раннім індикатором популярності та перспективного використання для розроблення програмних систем [302].

Щоб оптимізувати метод вимірювання популярності, ми провели ґрунтовні практичні дослідження. Отже, можемо об'єднати джерела даних із різними статистичними властивостями у такий спосіб, щоб робити усі значущі тенденції доступними, добре збалансованими для підрахунку загального бала популярності цієї технології зберігання та опрацювання інформаційних потоків даних. Згідно з запропонованою методикою можемо обробляти відсутні й помилкові дані. Водночас збір таких даних здійснюється в автоматичному режимі (рис. 1.6, 1.7).

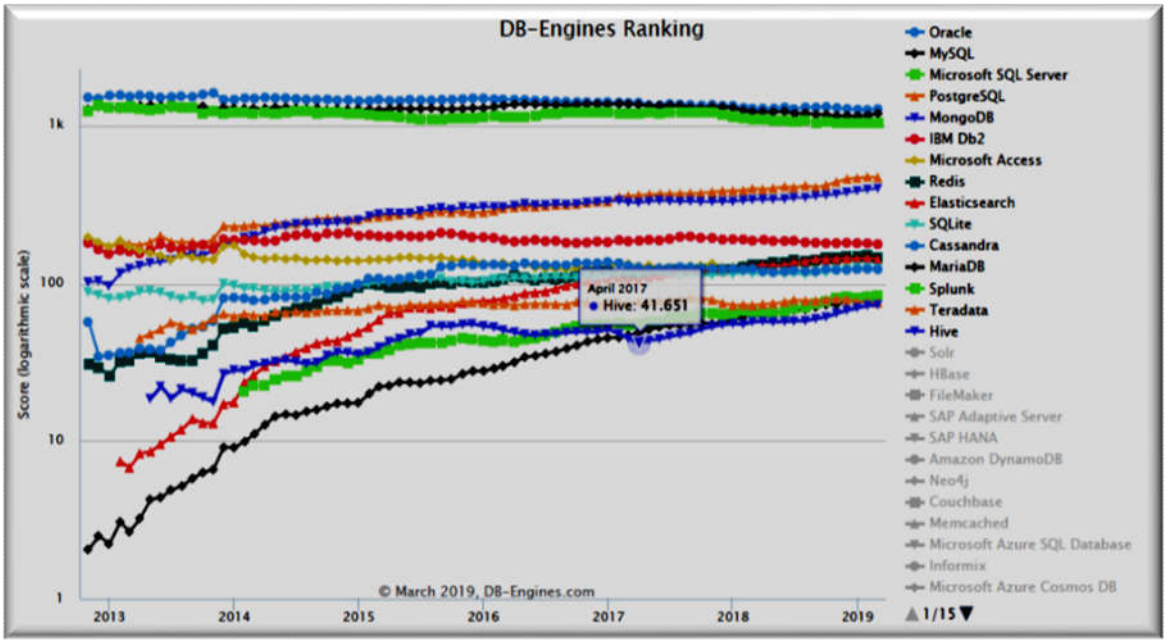


Рисунок 1.6– Тенденції популярності баз та банків даних у розробленні програмних систем (2013–2019)

Ґрунтуючись на дослідженнях із визначення популярності та перспективності використання систем управління базами та банками даних під час розроблення програмних систем із використанням хмарної технології, можна визначити параметри популярності:

1. Кількість згадок про систему управління базами або банками даних на Web-сайтах вимірюється як результуюча кількість запитів у пошукових системах. Зараз використовуються Google, Bing і Яндекс для визначення цього критерію оцінки. Для того, щоб розраховувати

тільки релевантні результати, ми шукаємо <ім'я системи> разом із терміном бази даних наприклад, «Oracle» та «бази і сховищ даних».

2. Загальний інтерес до системи управління базами або банками даних. Для цього параметру використаємо частоту запитів в Google Trends.

3. Періодичність технічних дискусій про систему управління базами або банками даних. Використовується низка публікацій щодо цього питання та кількість зацікавлених користувачів на відомому IT-ринку, які пов’язані з Q&A сайтами DBA Stack Exchange.

4. Кількість вакантних пропозицій щодо системи управління базами або банками даних. Використовується низка пропозицій на провідних пошукових системах про пошук дійсної роботи на Simply Hired.

5. Кількість профілів у професійних мережах, у яких згадується система управління базами або банками даних. Використовуються найбільш міжнародні популярні професійні мережі LinkedIn і Upwork.

6. Актуальність у соціальних мережах систем управління базами або банками даних. Визначаємо кількість твітів (Twitter), у яких згадується відповідна система опрацювання баз та банків даних.

Rank			DBMS	Database Model	Score		
Mar 2019	Feb 2019	Mar 2018			Mar 2019	Feb 2019	Mar 2018
1.	1.	1.	Oracle	Relational, Multi-model	1279.14	+15.12	-10.47
2.	2.	2.	MySQL	Relational, Multi-model	1198.25	+30.96	-30.62
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	1047.85	+7.79	-56.94
4.	4.	4.	PostgreSQL	Relational, Multi-model	469.81	-3.75	+70.46
5.	5.	5.	MongoDB	Document	401.34	+6.24	+60.82
6.	6.	6.	IBM Db2	Relational, Multi-model	177.20	-2.23	-9.47
7.	↑9.	7.	Microsoft Access	Relational	146.20	+2.18	+14.26
8.	↓7.	8.	Redis	Key-value, Multi-model	146.12	-3.32	+14.90
9.	↓8.	9.	Elasticsearch	Search engine, Multi-model	142.79	-2.46	+14.25
10.	10.	↑11.	SQLite	Relational	124.87	-1.29	+10.06

Рисунок 1.7 – Рейтинг популярності баз та банків даних у розробленні програмних систем (2018–2019)

Висновки до першого розділу

У розділі розглянуто сучасний стан використання інформаційних технологій та основні проблеми пов'язані з розробкою комп'ютерних програмних систем. Отже, відповідно до результатів дослідження з'ясовано, які є перспективні напрями розробки інноваційних інформаційних технологій для забезпечення конкурентоспроможності є на ринку України та за її межами, а також визначено, що інформаційні технології не залишаються у статичними, а перебувають у постійному динамічному розвитку.

Результатом цього дослідження є те, що інформаційні технології перебувають у динамічному стані розвитку і перманентної модифікації, оскільки до ринку IT-замовники ставлять дедалі складніші вимоги з інтерактивного й активного створення інформаційних структур для опрацювання та зберігання великих інформаційних потоків даних. Також акцентовано на розробленні інформаційних систем із використанням інноваційних інформаційних технологій для забезпечення бізнес-діяльності сучасного суспільства засобами програмних систем.

Для визначення перспективних напрямів розвитку інформаційних систем було проведено системний аналіз літературних джерел згідно з поставленими завданнями дослідження. Відповідно до результатів системного аналізу та проблем із розроблення прикладних програмних систем розділимо їх на декілька складових:

1. Системний аналіз життєвого циклу програмних технологій, для забезпечення правильного вибору цих технологій як зараз так і в перспективі.
2. Системний аналіз життєвого циклу баз та банків даних для розроблення програмних систем із використанням хмарної технології.
3. Аналіз та вибір хмарної технології для мінімізації бюджетних витрат на розробку і функціонування інформаційної системи.

РОЗДІЛ 2

ТЕОРЕТИЧНІ ОСНОВИ ГРУПОВОЇ ДИНАМІКИ У ФОРМУВАННІ КОМАНД РОЗРОБНИКІВ ПРОГРАМНИХ СИСТЕМ

2.1. Актуальність досліджень групової динаміки в сфері розроблення програмних систем

Системний аналіз досліджень і публікацій та теоретичне підґрунтя опрацювання цієї проблеми професійного спілкування відображено у працях вітчизняних і зарубіжних науковців: Л. Буєвої, Г. Бативощевої, С. Амеліної, Л. Барановської, О. Даниленко та ін. Когнітивні знання у галузі розробки програмних систем і формування професійних якостей інженерів із програмних систем проаналізували такі вчені: О. Кіриленко, О. Лучшева, Н. Падалко, Д. Щедролосьєв, Ф. Брукс, Г. Вейнберг, Н. Вірт та ін. [4, 59, 79]

Розроблення комп'ютерних програмних систем у сучасних реаліях є масовою професією, але водночас, на думку академіка А. П. Єршова, однією з найважчих. Складність її полягає у тому, що інженер-системотехнік має володіти здатністю першокласного математика до абстракції й логічного мислення разом із Едісонівським талантом споруджувати все що завгодно з нуля і одиниці[126, 141].

Складність розробки програмних систем не є основним чинником успіху. Визначальну роль в успішності програмного бізнес-проєкту відіграє команда розробників, її розмір, розташування і розподіл повноважень, а також нормативні обмеження та вимоги управління. Сьогодні не існує універсальної формули або технології для вирішення всіх проблем із розробки комп’ютерних програмних систем [181, 215].

Розроблення програмних систем – це складний технологічний процес із використанням різноманітних інформаційних технологій та команд розробників.

Команди розробників програмних систем працюють у різних умовах, і справді кожна команда перебуває в унікальній ситуації. Суть проблеми полягає у тому, що розробники повинні адаптувати

всі свої знання, підходи і фахові компетенції, щоб спрогнозувати ризики на які може натрапити команда. Проведений системний аналіз показав, що використання Agile software development технології під час розробки програмного забезпечення більш ефективне, а ніж розробка без використання Agile. Окрім того, застосування її методології Agile забезпечує вищий моральний дух команди, що водночас призводить до підвищення продуктивності праці та забезпечує посилення дисципліни серед персоналу команди [241, 243].

Зазвичай усі люди, котрі працюють в IT-сфері, – інтроверти, тобто зосереджені на своєму внутрішньому світі, черпають енергію «із себе» і витрачають її на події усередині себе. Потребують самоти для обробки інформації. Прагнуть спочатку скласти уявлення про об'єкт і тільки потім взаємодіяти з ним. Такі працівники не шукають широких контактів, хоча зовні можуть і не мати проблем у спілкуванні. У своєму дослідженні ми визначили типові ситуації, з якими зараз стикаються команди розробників комп’ютерних програмних систем. Не дивно, що у наслідок аналізу з'ясувалося, що команди розробників програмних систем працюють із дуже широким діапазоном ситуацій [3, 12, 23].

Залежно від бізнес-вимог розробники об’єднуються в невеликі команди у складі десятьох осіб чи менше або середні команди розробників програмних систем, мають від 11 до 50 осіб, проте є і дуже великі команди, які складаються з більш ніж 50–200 осіб. Малі команди переважно зосереджені в одному місці, проте використання сучасних комунікаційних технологій дає змогу створити команду, розподілених географічно або і інший спосіб. Деякі команди натрапляють на відносно прості проблеми розвитку, але здебільшого зі значними технічними і технологічними проблемами. Розмір команди, географічний та гендерний розподіл організаційних повноважень, технічної складності й нормативних проблем–ці фактори мають великий вплив на те, як працюють команди і як вони організують себе [25, 57, 142].

З огляду на достатньо великий спектр проблем, на які натрапляють групи розробників програмних систем, розуміємо, що команди повинні бути гнучкими у своїх підходах. Команда, що складається з п'яти людей, які працюють над проєктом із розробки програмної системи, буде працювати по-іншому, ніж команда з 20 чоловік, яка водночас працює по-іншому, ніж команда з 200 чоловік. Команда, яка перебуває в одному офісі, буде працювати по-іншому, ніж команда,

яка розсіюється в різних офісах, яка водночас працює по-іншому, ніж команда члени якої розкидані географічно [26, 70, 149].

Різні команди у різноманітних ситуаціях вимагають різних організаційних структур, різноманітного розбиття програмних модулів для забезпечення бізнес-процесу і різного оснащення конфігурацій. Групи більшого розміру вимагають більше спілкування та організаційних витрат, отже, мають більший ризик, ніж менші команди. Координація у межах невеликої Agile-команди вирішується завдяки щоденним stand-up meeting (раз на два тижні), детальному плануванню сесії з розробки програмної системи. Середній же команді можуть знадобитися два рівні stand-up meeting – «сутичка» і «scrum of scrum-зборів», а також частіші сесії планування [27, 156, 163, 169].

Відбір і формування команди фахівців відбуватимуться у продовж цього процесу розробки програмної системи і суттєво впливають на неї, тому у команді цінують як спеціальні інженерні навики, пов'язані з розробкою програмних систем, так і загальнолюдські якості, які під впливом специфіки професійної діяльності стають професійними [195, 216].

Варіативними гранями своєї психіки люди реагують на оточення. Тому найбільш важливим є визначення соціальної психології як науки, що намагається зрозуміти і пояснити, який вплив домінує над думками, почуттями і індивідуальною поведінкою насправді, а також уявна або передбачувана присутність інших. Реакція людини на присутність інших проявляється у вигляді спектра різноманітних ефектів [196, 213, 236].

Групові ефекти – це механізми функціонування групи, за допомогою яких здійснюються групові процеси і досягаються групові стани. Вони є засобами, що забезпечують інтеграцію індивідуальних дій у спільній груповій діяльності і спілкуванні (рис. 2.1).

Члени команд розробників, які об’єднуються у групу неодмінно зіштовхуються з груповими ефектами. Розглянемо, з якими ефектами стикаються групи програмістів – розробників системи під час їхнього існування:

1. Ефект соціальної фасилітації – це організація процесу колективного розв'язання проблем у групі, яка керується фасилітатором (ведучим, керівником). Це одночасно процес та сукупність навичок, які дають змогу ефективно організовувати обговорення складної проблеми без втрат часу та за короткий термін виконати усі заплановані дії із максимальним залученням учасників процесу. Фасилітація від-

різняється від простого управління тим, що вона не має директивного характеру. Якщо в традиційних формах управління суб'єкт змушує учасників групи виконувати власні інструкції та накази, то у випадку з фасилітацією її суб'єкт має поєднувати у собі ознаки керівника, лідера та учасника процесу створення програмних систем.

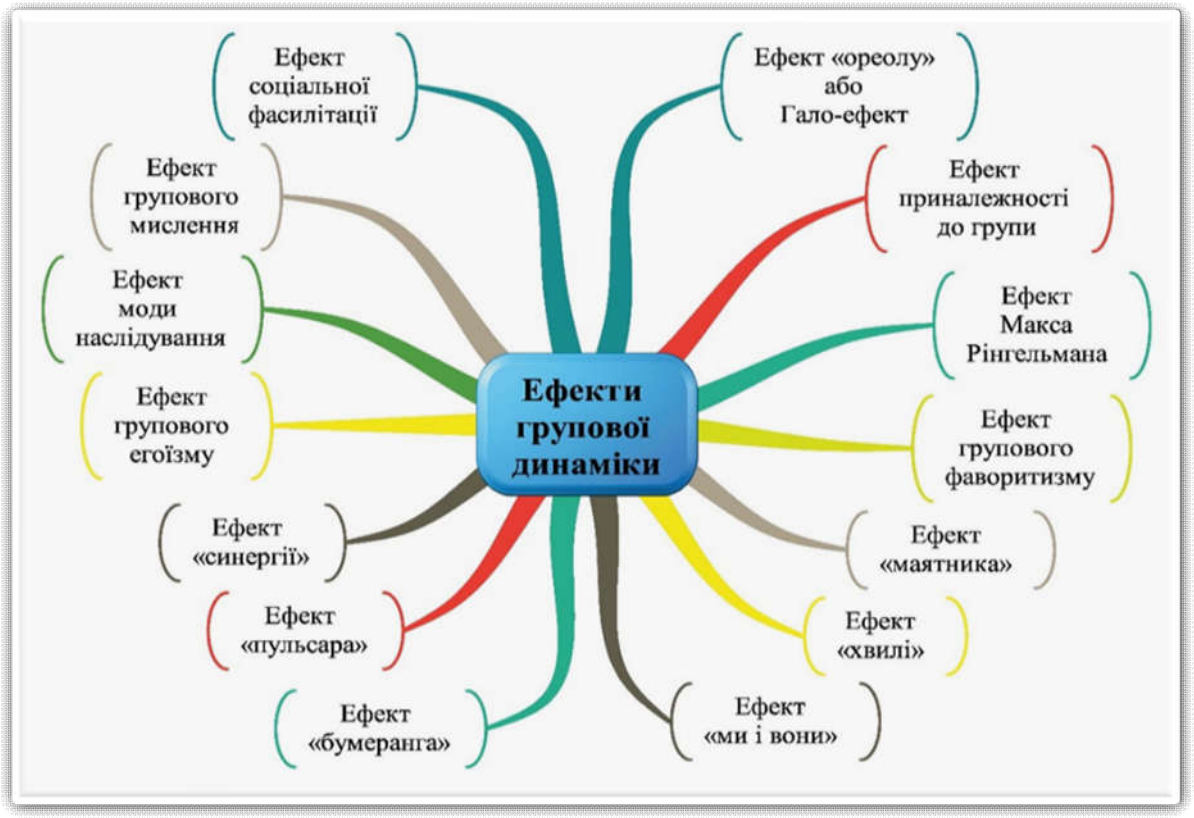


Рисунок 2.1 – Модель ефектів групової динаміки

2. *Ефект приналежності до групи* – один із соціальних інстинктів, властивих людям, полягає в ототожненні себе з групою і прагненні оцінити її роботу позитивно, підтримуючи високий статус групи й одночасно власну самооцінку (Е. Мак-Дауголл, 1908 р.).

3. *Ефект Макса Рінгельмана* свідчить, що із збільшенням кількості осіб групи відбувається зменшення середнього індивідуального внеску у групову роботу. Він визначає, що продуктивність групи не перевищує половини суми продуктивності її осіб. Дослідження Рінгельмана засвідчило, що члени групи фактично менш мотивовані і докладають менше зусиль підчас виконання спільних дій, ніж індивідуальних. Науковець порівнював, наприклад, підняття ваги разом із групою та окремими особами.

4. *Ефект синергії* – це виникнення додаткової когнітивної інтелектуальної енергії під час об'єднання людей у цілісну групу, що вті-

люється в груповому результаті, які переважає суму індивідуальних результатів (В. М. Бехтерєв, М. М. Ланге).

5. Ефект групового мислення – це специфічний спосіб мислення, за якого в об'єднанні групі домінує пошук згоди, що підпорядковує собі реалістичну оцінку можливих альтернативних дій. Відкриття цього феномену, як і запровадження терміна «групове мислення» (groupthink), належить Ірвінові Дженісу. Описаний ефект виникає тоді, коли критерієм істинності слугує узгоджена позиція групи, яка протиставляється думці окремої людини. Коли ж члени групи отримують загрозу виникнення розбіжностей у позиціях, думках, суперечках та конфліктів, вони намагаються зменшити груповий когнітивний дисонанс і усунути негативні почуття, які при цьому виникають, та водночас знайти рішення, дане рішення може бути недостатньо продуманим і раціональним з погляду кожного члена команди розробників програмних систем. Якщо команда втягується у такі стратегії прийняття рішень, пошуки консенсусу стають настільки важливими, що члени команди відмовляються добровільно від будь-яких сумнівів і можливостей подивитися на завдання по новому [75]. Окремі члени команди можуть ставати також так званими груповими наглядачами, які зайняті тим, що відстежують і жорстко карають будь-яке інакомислення.

6. Ефект моди наслідування. Наслідування – один з основних механізмів групової інтеграції. Під час групової взаємодії члени групи формують спільні еталони, стереотипи поведінки, дотримання яких підкреслює їхню єдність та зміцнює їхнє членство у групі. Члени таких груп створюють певні усталені норми щодо зовнішнього вигляду уніформа у військових, ділові костюми бізнесменів, білі халати лікарів. Така групова уніформа, іноді не встановлена офіційно, показує оточенню, до якої групи людина належить, якими нормами та правилами регулюється її поведінка. Люди схильні наслідувати приклад людини, чимось схожої на себе, більше ніж на схожі. Ефект наслідування лежить в основі будь-якого навчання і сприяє адаптації людей одне до одного, узгодженості їхніх дій, підготовленості до вирішення групових задач.

7. Ефект ореолу, або Гало-ефект – це результат дії загального враження про що-небудь (явище, людину, речі) на сприйняття його деяких особливостей. Прикладом може слугувати враження, що у людей з привабливою зовнішністю великі розумові здібності. Першим навів експериментальні докази існування гало-ефекту Едвард Лі

Торндайк. В управлінні персоналом гало-ефект – це джерело помилок в оцінці особи, адже спостерігач користується лише першим враженням або рисою, що запам'ятовується, в оцінці індивідуальності. Ефект ореолу у сфері бізнес-аналізу описаний в однойменній книзі Філа Розенцвейга.

8. Ефект групового фаворитизму – це тенденція сприяти членам своєї групи на відміну від членів чужої групи. Цей ефект слугує розподільним механізмом між людьми, що сприймаються як свої і чужі. Ефект групового фаворитизму сильніше проявляється тоді, коли для групи дуже важливі критерії порівняння результатів діяльності й особливості стосунків з іншими групами. Коли вони конкурують одна з одною, утворюються можливості для однозначного порівняння груп. Коли членство у групі більш важливе, ніж міжособистісна подібність, тоді віддають перевагу «своїм», навіть якщо «чужі» є подібними за своїми особистими якостями, інтересами, поглядами. Члени групи також схильні пояснювати успіх своєї групи внутрішньогруповими чинниками, а можливу невдачу – зовнішніми. Тож якщо група успішна у своїй діяльності, вона завдячує самій собі (своєму керівництву, клімату, здібностям її членів). Коли ж група зазнає поразки (невдачі), то шукає винуватців за межами групи або скидає провину на інші групи.

9. Ефект групового егоїзму – це спрямованість групових інтересів, цілей і норм поведінки проти інтересів, цілей і норм окремих груп чи всього суспільства. Цілі групи водночас досягаються через протидію інтересам членів інших груп, нехтування суспільними інтересами. Груповий егоїзм проявляється тоді, коли цілі та цінності групи стають важливішими за суспільні цінності, коли поступаються інтересами окремої людини задля стабільного існування групи. У таких випадках члена команди приносять в жертву цілісності групи, її повністю підпорядковують вимогам та стандартам групової поведінки. Цей ефект має надзвичайно негативні наслідки для групи загалом, її подальшої життєдіяльності та долі її окремих членів.

10. Ефект маятника – це циклічне чергування емоційних станів сценічного та астенічного характеру, інтенсивність і тривалість яких залежить від діяльності групи. Експериментально емоційні потенціали групи досліджував О. М. Лутошкін. Емоційні цикли групи залежать від чинників дня тижня та періоду доби.

11. Ефект хвилі – це поширення у групі ідей, цілей, норм і цінностей, коли вони відповідають потребам та інтересам людей, а не

суперечать їм. Окрема людина ділиться новою ідеєю зі своїм найближчим оточенням, її доповнюють та розвивають члени групи. Ідея починає поширюватися серед інших членів групи, здійснюється її групова оцінка та обговорення, ідея захоплює дедалі все більше людей. Це можливо лише тоді, коли нова ідея відповідає потребам та інтересам людей, а не суперечить їм. Якщо ідея відповідає інтересам людей і розвивають її, то ефект хвилі посилюється. Якщо ідея суперечить інтересам людей, то хвиля затухає.

12. Ефект пульсара – це зміна групової активності залежно від різних стимулів. Групова активність розгортається як цикл: оптимальна активність, необхідна для нормальної роботи групи; підйом активності; спад активності; повернення до оптимального рівня активності. Розгортання цього циклу залежить від зовнішніх (отримання групою термінового завдання) і внутрішніх (прагнення членів групи вирішити проблему) стимулів. Відповідно до ефекту пульсара активність групи різко підвищується на початку діяльності, а коли завдання вирішене, настає її спад. Потім рівень активності знову піднімається до оптимального рівня, необхідного для нормальної злагодженої роботи групи.

13. Ефект бумеранга – невизнання істинною сприйнятої інформації, дотримування попередньої установки або формування нової оцінки подій чи особи, протилежного змісту, ніж повідомлена інформація. Вперше досліджений у діяльності засобів масової інформації, цей ефект полягає у тому, що людина, яка сприймає інформацію, не визнає її істинною, а продовжує дотримуватися попередньої установки або формує нову оцінку подій чи особи, протилежного змісту, ніж та інформація, яку людині повідомили. Ефект бумеранга виникає при повідомленні суперечливої інформації або при взаємодії людей, коли агресивні дії однієї особи, спрямовані проти іншої, врешті-решт діють проти того, хто здійснює ці дії або негативно висловлюється. В умовах групи люди більш прихильні до спокійної людини, ніж до її агресивного суперника.

14. Ефект ми і вони – це почуття належності до групи (ефект «ми») і, відповідно, відстороненості, відокремлення від інших (ефект «вони»). Ефект належності до групи має два окремі ефекти – емоційної підтримки та залучення. Ефект залучення полягає у тому, що член групи відчуває себе залучення до проблем, справ, успіхів чи невдач тієї групи, до якої він реально належить чи суб'єктивно себе до неї приєднує, відчуває відповідальність за результати групи. Ефект емо-

ційної підтримки проявляється у тому, що член групи очікує емоційної та реальної підтримки, співчуття, допомоги з боку інших членів групи. Якщо член групи не отримує підтримки, у нього руйнується почуття «ми» – почуття належності до групи і виникає почуття «вони», тобто він здатен сприймати свою групу як чужинців, що не поділяють його інтересів і турбот. Ефект «ми» є психологічним механізмом функціонування групи. Гіперболізація почуття «ми» призводить до переоцінки своїх можливостей і переваг, до відриву від інших груп, а також групового егоїзму. Водночас недостатня розвиненість почуття «ми» призводить до втрати почуття ціннісно-орієнтаційної єдності групи.

Отже, учасники групи повинні подолати внутрішні суперечності, пройти через конфлікти перш, ніж сформується дійсно злагоджений колектив зі створеними загальнокомандними цілями. Можливе виникнення суперництва, суперечок, оборонної позиції, неминучі труднощі або невдачі можуть спричинювати конфлікти, «пошук винних». Лідеру на цьому етапі важливо забезпечити відкриту комунікацію у команді – конфлікти не потрібно ховати або розривати. Суперечки необхідно вирішувати спокійно, терпляче і ретельно. На зміну битві амбіцій обов’язково прийде продуктивна співпраця, чіткий розподіл праці, зникає дублювання функцій. Лідер перестає перебувати у постійного авралу, робота по побудові команди на цьому етапі – це вже не гасіння пожежі, а скрупульозна праця з відпрацювання загальних норм і правил [239, 244, 246].

Обов'язково повинні бути сформульовані і впроваджені механізми результативного перегляду правил, які втратили свою актуальність або ефективність. І настає той етап, коли команда працює продуктивно, відчувається високий командний дух, люди добре знають одне одного й уміють спільно використати позитивні сторони колег [60, 76]. Високий рівень довір'я – це кращий період для розкриття індивідуальних талантів. Люди хочуть і можуть удосконалюватися, вони найбільше зацікавлені у професійному зростанні.

2.2. Методи та засоби групової динаміки формування команд

Наслідки безперервних технологічних удосконалень, поліпшення звітності та зміни у професійному житті вимагають нових компетен-

цій від сучасних працівників. Однак визнання важливості змін автоматично не приводить до впровадження успішних інновацій, тому головною метою дослідження впливів групової динаміки є аналіз наукових публікацій з метою вироблення методик для запровадження у розробку програмних систем [24, 31, 43].

Люди найважливіший ресурс будь-якого підприємства. Успішність виконання будь-якого бізнес-проекту безпосередньо залежить від членів команди, залучених до процесу розробки програмних систем. Команда та її члени інтелектуальний капітал будь-якої організації. Для успішного управління цими ресурсами необхідні спеціальні знання та навички. Ефективне управління командою розробників програмних систем визначає успішність виконання проєкту.

Керівники проєктами розробки програмних систем часто займаються групами людей, які повинні виконувати певну роботу, наприклад специфікацію вимог, проєктування, конструювання, алгоритмізацію процесів, побудову та тестування програмного забезпечення для комп’ютерних систем [231, 232].

Розробка програмних систем починається із формування команд методом групової динаміки. Насамперед, управління проєктами з розробки програмних систем пояснюється складністю та комплексним підходом до такої інтелектуальної роботи. Складний проєкт, що складається з декількох завдань, не може бути реалізований однією людиною, і навіть якщо він може бути реалізований, цей процес буде займати занадто багато часу для завершення. Тому природним рішенням цієї проблеми є організація команди з фахівців, які можуть виконувати необхідний обсяг робіт у най коротший термін [235, 237]. У цій ситуації менеджер проєкту стикається з проблемою управління роботою групи розробників програмних систем. Це особливо яскраво виражено для розробки програмних систем як робота білих комірців.

Концепція групи має кілька аспектів визначення. Групою можна вважати двох осіб або більше, пов'язаних між собою у межах як виробничих так, і соціальних стосунків. Таке визначення підкреслює важливість комунікації між членами групи та ключову роль, яку відіграють взаємозв’язки всередині групи. K. Lewin визначив процеси як групи як окремі особи діють та реагують на динаміку групових умов. Групова динаміка розглядає різні аспекти взаємодії членів групи. Ця наука має багатопрофільний характер, він поєднує переваги психології, антропології, бізнес-менеджменту, соціології, психічного здоров'я, політології та комунікаційних досліджень [137].

Груповий процес стосується природи стосунків між двома особами, які взаємодіють один з одним, як це визначив I. Yalom. Групова взаємодія не має обов'язково зосереджуватись на словесній взаємодії, індивідуальному способі спілкування, змісті та характеру взаємодії. Тепер очевидно, що такі взаємодії і комунікації повинні управлятися належним чином.

Розроблення програмних систем визначається комп'ютерним суспільством IEEE як «застосування систематичного, дисциплінованого, кількісного підходу до розробки, експлуатації та супроводу програмного забезпечення у системах, тобто застосування обчислювальної техніки для розробки програмного систем» і як наслідок дослідження з цих підходів [234, 238, 242]. Найчастіше всі серйозні розробки програмних систем, незалежно від величини великі або малі, передбачають, що їхня реалізація повинна виконуватися у межах робочих груп проєктів.

За різними моделями життєвого циклу програмного забезпечення можна виділити загальні процеси розроблення програмних систем. Вони містять вимоги до техніки, програмного забезпечення проєктування, конструювання, внутрішньої і зовнішньої взаємодії програмних компонентів, систем прийому та передачі інформаційних даних, систем зберігання та опрацювання даних, систем захисту як програмного забезпечення, так і інформаційного наповнення системи, а також випробування й технічний супровід. Усі ці основні процеси вимагають від членів команди дотримуватись методів групової динаміки і працювати як годинниковий механізм.

Значна частка розроблених програмних систем у світовому вимірі розроблена командами різних розмірів. Розробка проєктів програмних систем часто досить складна, тому проєкт розбивають на спрінти (Sprint), із розрахунку 1-2 тижні на виконання над ним, які поступово або паралельно опрацьовують команди розробників програмного забезпечення. Вони формуються у групи з 5-10 учасників, які тісно взаємодіють між собою для успішного вирішення поставленого завдання, щоб вирішити деякі конкретні проблеми і створювати готову підсистему, яка є частиною усієї складної програмної системи. Організація та управління розробки програмної системи охоплює деякі проблеми, пов'язані з організацією саме ефективних робочих груп методами групової динаміки, які можуть виконувати поставлене завдання у встановлений час та з обмеженими ресурсами [248].

Найбільш популярними методами управління груповою динамікою у розробників програмних систем є вирівнювання участі кожного із членів команди розробників, підбиття підсумків, заклик до консенсусу, проходження буфера обміну, цензура та виключення. Такі методи групового обговорення містять мозковий штурм, турніри, цілу групу та невеликі групові обговорення і їхні різноманітні форми, такі як стояче обговорення, засідання та інші.

Ці методи спрямовані на вдосконалення роботи команди та створення командної згуртованості. Отже, такі методи групової динаміки можуть бути реалізовані у проєктах із розробки програмних систем. Проте у нашому суспільстві зростають темпи потреб у глобально-розподілених проєктах які відстежуються на всіх ринках із розробки програмних систем. Це зумовлює розвиток нової категорії робочих груп та методів їхнього управління, а саме групової динаміки. Ці методи управління передбачають використання сучасних електронних способів зв'язку на базі засобів Інтернет та мобільного зв’язку [250-254]. Крім того, поряд із програмним забезпеченням існують інженерні процеси, які часто потребують вироблення ефективних рішень, – це окремий клас прийомів, методів та засобів, передбачений спеціальним класом програмно технічних рішень.

У наукових публікаціях, пов'язаних із методами групової динаміки, велика увага приділяється опису методів, але небагато науковців акцентує увагу на аналізі їхнього впливу на робочу групу, що працює як одне ціле (команда) в галузі розроблення програмних систем. Ще один момент полягає у тому, що рідко аналізується вплив розміру і структури команди за вибором реалізованих методик та їхній вплив на досягнення позитивних результатів [255-257].

Ключовим завданням дослідження є:

1. Дослідити методи групової динаміки та їхній вплив на команди в розробленні програмних систем, здійснивши системний огляд джерел.

2. Визначити проблеми, які стоять перед командами, що формуються на базі методів групової динаміки під час розробленні програмних систем.

3. Дослідити, які з існуючих методів групової динаміки фактично використовуються у промисловості при розробленні програмних систем.

З метою досягнення поставленої мети та завдань окреслимо такі питання, які мають бути досліджені у нашій праці:

1. RQ1. Які методи використовуються для підтримки групової динаміки у групах із розроблення програмних систем?

2. RQ2. У чому полягає вплив методів групової динаміки на групи із розроблення програмних систем?

3. RQ2.1. Як динамічні методи групи оцінюються у групі із розроблення програмних систем?

4. RQ2.2. Які виникають проблеми у реалізації методів групової динаміки при розробленні програмних систем та як ці проблеми пом'якшують?

5. RQ3. Які з існуючих методів фактично використовуються в промисловості і який вони мають вплив на розробку програмних систем?

Очікуваний результат дослідження дає змогу оцінити важливість технологій групової динаміки у розробленні програмних систем для того, щоб практикуючі фірми промисловості взяли результати даних досліджень, а також оцінили важливість групової динаміки та її виклики, які сприяють пом'якшенню конфліктних наслідків, що виникають у процесі функціонування групи.

Для того, щоб фахово і коректно відповісти на питання RQ1 і RQ2, потрібно провести детальний огляд наукової літератури. Щоб забезпечити коректність доведення нашої гіпотези, ми пропонуємо використовувати тільки кращі дослідження з обраної теми. Щоб провести та дослідити наукові статті, матеріали конференцій, довідники та книги, було запропоновано використовувати методологію SLR (systematic literature review) пошуку. Це дає змогу збирати відповідні дослідження та проводити вилучення тих даних, які не відповідають заданим критеріям. Використання SLR забезпечує кращий ефект для збору наявних знань та надає певні гарантії, що ми не пропустимо жодної відповідної статті, у випадку якщо будемо коректно дотримуватися цієї методології [138].

На практиці існують різні види літературних оглядів. Найбільш поширеним є систематичне відображення досліджень. У ситуації, коли дуже мало доказів або тема дуже широка, цей вид огляду є найбільш вдалим вибором. Запропонована методологія дає змогу здійснити науковий огляд публікацій у вільній формі на основі статей, знайдених у відкритому доступі.

Методологія SLR забезпечує більш формалізований тип огляду наукових джерел. Це методично строгий огляд результатів досліджень. Він визначає окремі етапи і процедури, які необхідно викона-

ти для того, щоб можливо було отримати достовірні результати. SLR призначена для підтримки розвитку наукової доказової бази та формування керівних принципів для практиків.

Означені наукові питання у дослідженні вимагають використання методології SLR, яка забезпечує структуровану й обґрунтовану процедуру відбору та аналізу первинних документів обробки даних. Ми не можемо застосовувати традиційний огляд літератури, тому що він здебільшого використовується для узагальненої теми, хоча у нас є певні питання щодо методів групової динаміки та їхнього впливу на виконання поставлених завдань із розроблення програмних систем.

Методологія SLR дає змогу мінімізувати масиви виділення первинних досліджень групової динаміки для розробки програмних систем. Отже, висновки, зроблені на основі методології SLR, більш надійні і посилені за допомогою коректного видобування даних. Під час опрацювання наукових масивів публікацій ми будемо шукати, збирати і підсумовувати докази впливу групової динаміки на команди із розроблення програмних систем. Цю методологію було обрано в якості методу дослідження, оскільки вона забезпечує належний спосіб видобування та опрацювання відповідей від фахівців із практичними навиками з усього світу. А це і є однією з наших головних цілей, яка полягає в тому, що ознайомитися з фактичною ситуацією щодо побудови команд в галузі (рис. 2.2).

Також обстеження дає реальні результати, які виявилися надійними та достовірними під час статистичного аналізу, це забезпечило автоматизовану обробка результатів, що робить цей метод швидким, з низькою вартістю у той же час надійним.

Методологія SLR також була використана для вивчення RQ3, а саме в яких промислових задачах використовувалися методики групової динаміки з розроблення програмних систем.

Проводячи пошукові дослідження на предмет використання групової динаміки у промисловості, ми можемо дослідити, як наявна методологія фактично реалізується й оцінюється в розробленні програмних систем. Системний аналіз огляду літературних джерел згідно з обраним об’єктом тематики є вторинним дослідженням, хоча окремі пошукові запити дають змогу проводити систематичний їх огляд [259-261, 280].

SLR містить кілька етапів, які повинні бути реалізовані під час нашого дослідження (рис. 2.3).

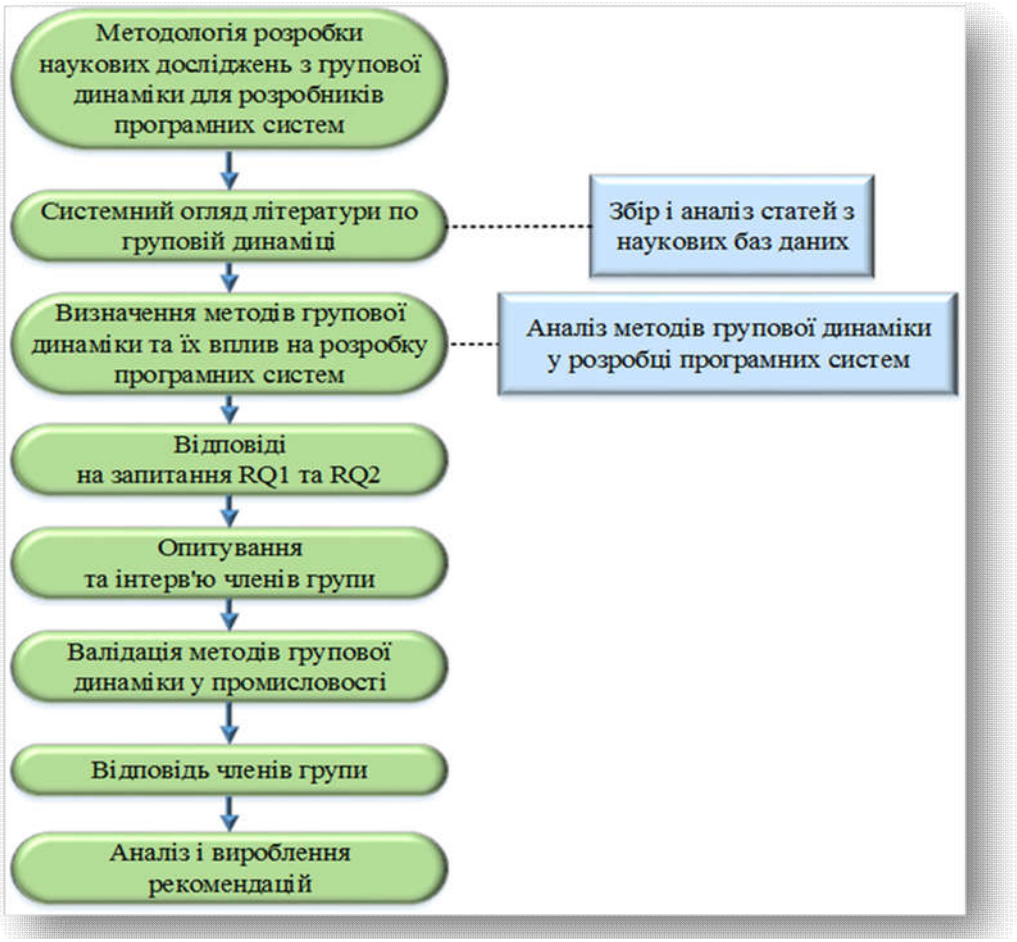


Рисунок 2.2 – Методологія аналізу впливу групової динаміки на команди

Насамперед необхідно спланувати усі види діяльності нашого дослідження з пошуку та опрацювання інформації щодо розроблення програмних систем. Проведення огляду об'єднує етапи генерації пошуку документів та визначення критеріїв відбору, колекції первинного дослідження, а також проведення оцінки якості, видобування даних і їхнього синтезу.

Метою цього системного огляду літератури є пошук, вивчення та аналіз наукових публікацій, присвячених проблемам наявних методик підтримки групової динаміки при формуванні команди в розробленні програмних систем, а також їхній вплив на команди, що працюють в інженерії програмного забезпечення. Оцінка результатів реалізації методики пошуку RLS є метою RQ2.1 «Як динамічні методи групи оцінюються у групі з розроблення програмних систем»? Виявлені методики мають свої власні заходи з підвищення ефективності, наприклад, своєчасність, щільність дефектів, ступінь заходів та угоди. Ці заходи пов'язані з виконанням команди, витратам на проєкт із розроблення програмної системи, оцінкою ризику прийняття конкретного рішення.

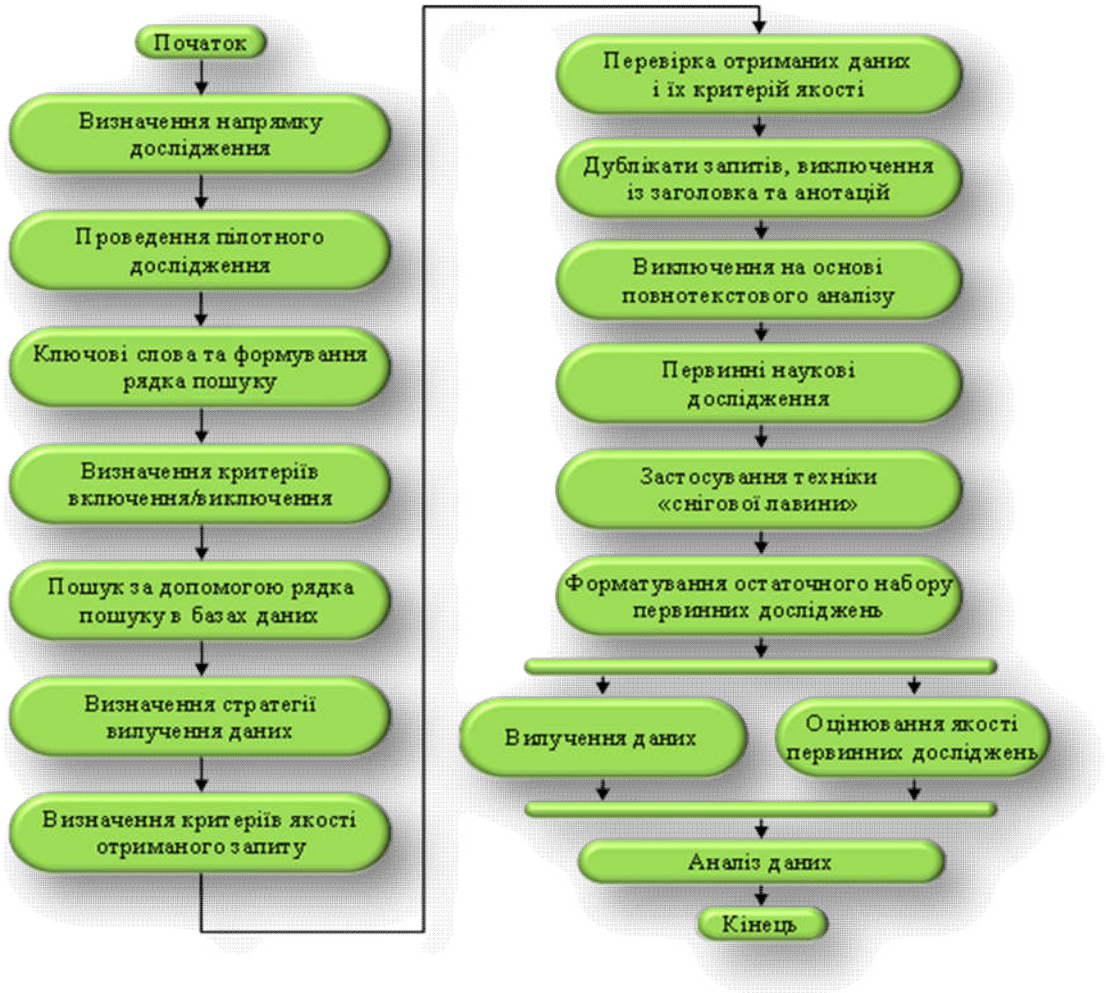


Рисунок 2.3 – Методологія опрацювання наукової літератури у базах даних

Для вирішення питання RQ2.2 «Які виникають проблеми у реалізації методів групової динаміки при розробці програмних систем та як ці проблеми пом'якшують? Необхідно враховувати реалізацію методів групової динаміки. Озвучені відкриті питання, а також ці питання повинні бути визначені й проаналізовані в даній науковій роботі.

2.3. Теоретичні основи групової динаміки при формуванні команди розробників програмних систем

Групова динаміка належить до взаємозв’язків між людьми, які працюють разом у робочій групі над розробкою програмної системи. Теоретичні методи групової динаміки можна вивчати у різних умовах: бізнес-налаштування добровольців, налаштування класу та соці-

альні налаштування. Якщо у будь-який час є три або більше людей, які працюють разом, то вони взаємодіють за методиками групової динаміки [306].

Іншим визначенням групової динаміки є ступінь, в якій особи робочої групи беруть активно участь у прийнятті рішень, діляться своїми ідеями про завдання, мотивуючи інших співробітників. Динаміка команди і групи важлива у будь-якій галузі та на будь-якому рівні, важливість цієї динаміки відіграє ключову роль у загальному успіху програмної організації. Групова динаміка може належати до розвитку групи та руху в часі до її мети, зумовлених взаємодією і спілкуванням членів групи й зовнішнього впливу. Розглядається сукупність соціально-психологічних процесів і явищ протягом усього життєвого циклу робочої групи [248, 263]. Групова динаміка охоплює всі етапи роботи групи, а саме розвиток, починаючи з її створення, функціонування, розвитку стагнації, регресу і розпаду. Відповідно до наведених визначень, у цьому досліджені розглядається групова динаміка як принципи, за якими люди працюють разом для досягнення загальної поставленої мети.

Існують різні типи груп, наприклад:

– сім'ї; соціальні групи (більш великі і більш формально організовані, і членство, як правило, коротше за тривалістю та з меншим емоційним залученням;

– колективи (учасники виступають у схожих, а іноді і незвичайних категоріях, агрегування осіб, які схожі один на одного в деяких напрямах).

Ми розглядаємо групи співробітників із розроблення програмних систем компанії, які фактично представляють тимчасово створенні колективи. У такому випадку життєвий цикл групи може бути представлений класичною моделлю Bruce Tuckman’s. Ця модель включає у себе такі етапи: формування, штурмування, нормування, виконання та відлагодження. Команди учасників знайомляться один з одним на стадії формування групи. Під час цього етапу учасників хвилює більше враження, яке вони створюють у своїх колег по групі, а не сам проєкт. Це найважчий час для групи, коли команді доводиться переходити зі стану «як є» до стану «бути командою» [184-186].

Стан узгодженості характеризується консенсусом та довірою між членами робочої групи. Під час цього етапу команда врегульовує свої стосунки та очікування. Інше поняття, дуже близьке до групової динаміки, – це команда і командна робота. Існує невелика різниця між гру-

пами і командами з погляду управління людськими ресурсами. Команда – це широкий асортимент людських агрегатів [22, 180, 188]. Команди схожі на групи і мають основні характеристики будь-якої групи: взаємодія, цілі, взаємозалежність, структура, єдність. Проте інтенсивність кожного з цих атрибутів у командах різна (табл. 2.1), і ми підсумовуємо тринадцять відмінностей між групами і командами [17].

Таблиця 2.1 – Порівняння груп та команд

Порівняння критерії	Група	Команда
Мета	Вузька проблема без знання глобальних цілей	Чіткі і зрозумілі цілі та стратегії їх досягнення
Участь	Виконання обов'язків та інструкції	Активна позиція, заснована на результатах мотивація. Обмін результатами незалежно від особистого самодіяльність
Рольова структура	Суворий розподіл ролей і обов'язків, взаємозалежні	Розподіл компетенцій, гнучка зміна ролей, щільно пов'язані відносини
Управління	Адміністративне управління, наявність керівника	Лідерство на основі компетенцій та довіри, допомога й підтримка
Прийняття рішень	Рішення, приймаються більшістю голосів або керівником групи	Ефективні процедури прийняття рішень
Конфлікт, вирішення	Ігнорування конфліктів	Прийняття конфліктів, викликаних інтелектуальними компетенціями, ефективне вирішення
Взаємодія	Цілеспрямованість, близькість, уникнення критики	Цілеспрямованість і підтримання відносин. Довіра, свобода, свобода ініціативи
Комуніка-ційні	Через формального лідера	Відкритість, взаємна повага
Творчість	Працюємо за правилами, підтримка стереотипів	Гнучкість і адаптація, постійне удосконалення та зростання компетентності, підтримка творчого потенціалу

Також є і різні типи команд. Можна виділити менеджмент, проект, консалтинг, сервісні, виробничі та ініціативні групи. Мета управлінської команди – це прийняття рішень у повсякденній діяльності та виробництві, а також встановлення цілей на майбутнє організації [187]. Команда з розроблення програмної системи об'єднує фахівців із різними знаннями, які працюють разом над створенням інтелектуального рішення. Консультативні групи представлені головним чином експертами групи та керівними комітетів. Виробничі бригади відповідають за практичну реалізацію поставленого завдання, а ініціативні групи – це високо скоординовані групи людей, які мають єдину мету та спрямовані генерувати свої ідеї на певний результат.

Командна робота – це психологічна, поведінкова та розумова робота, яку виконують усі члени команди, коли вони співпрацюють між собою над різними завданнями та підзавданями, які вони повинні вчасно виконати, щоб досягти бажаної мети [22]. Ідеальна команда має низку певних характеристик, які належать до трьох областей: це їх зворотний зв'язок і комунікативна поведінка, їх поведінка та люб’язність, а також способи їх наближення до завдань і проблем [34].

Життєвий цикл розроблення програмної систем у команді аналогічний такому ж, як і в групі. Управління групою та робота у команді взагалі вимагає розгляду наступних процесів з питань: лідерство, прийняття групових рішень і пошук спільної думки, формування ролей і відповідних функцій, групова згуртованість, вирішення конфліктів, індивідуальна поведінка в групі, індивідуальність та ефективність команди.

Усі ці фактори пояснюють психологічні зміни у групі, комунікації всередині та результати її роботи. Уся діяльність керівників управління повинна ґрунтуватися на принципах послідовності, поваги, наполегливості та чесності. Щодо проєктів із розроблення програмного забезпечення, то, швидше за все, ми повинні вивчати команди та командну роботу.

Але оскільки групова динаміка стала дуже поширеним поняттям, ми включаємо їх до уваги під час проведення досліджень за спеціальною технологією та їхній вплив на перебіг і виконання розроблення програмної системи.

2.3.1. Основи управління групами у галузі розроблення програмних систем

Інженерія програмних систем є складною багатопрофільною наукою, яка розглядає програмне забезпечення як вагомий етап у розробленні комп'ютерних програмних систем групами фахівців [35]. Як було сказано вище, всі основні процеси розробки комп'ютерних програмних систем вимагають об'єднаної та цілеспрямованої роботи у команді. Організація ефективної команди – комплексне завдання із залучення до колективу розробників найкращих фахівців, які мають достатню кваліфікацію, знання, навички та досвід для виконання поставленого завдання [36]. Крім того, такий фахівець повинен бути достатньо комунікабельний і комунікативний, щоб працювати разом у команді, ділитися результатами та спільними рішеннями [39]. Кожен член команди повинен думати про загальні інтереси та цілі команди. Сучасні підприємства, які розробляють комп'ютерні програмні системи, зазвичай працюють за програмним методологіями розробки. Це може бути модель водоспаду, інкрементна, V-подібна, спіральна або Agile-методологія. Методологія, заснована на технології водоспаду або каскадної моделі, розглядаються «основні процеси діяльності, специфікації розробки, перевірки, еволюції та представляє їх як окремі процеси і етапи наприклад, специфікація вимог, розробка програмних систем, впровадження, тестування» [40, 50, 51, 54].

Приклади реалізації такої методології охоплюють Rational Unified Process (RUP), Microsoft Solutions Framework (MSF), які є «величезними» і документально інтенсивними методологіями. RUP визначається як ітеративна, архітектурно-орієнтована розробка програмних систем управління бізнес-логікою виробничого процесу, що використовується для автоматизованого опрацювання [46]. Серед основних позитивних принципів RUP ми можемо підкреслити ідею про те, що вона вміє працювати разом як одне ціле у команді [42]. Успіх проєкту з використанням методології RUP досягнутий шляхом застосування кращих практик із розроблення комп'ютерних програмних систем, управління та розвитку разом з принципами командної роботи. MSF надає узгоджений набір концепцій, моделей і правил для розробки та розгортання інформаційних систем на базі технологій і засобів Microsoft [48]. Модель – проєктна група відповідно до MSF ґрунтується на звіті про рівність ролей. Кожна роль у кластері – це група

професіоналів, фахівців тієї ж ролі, які мають унікальну думку про проєкт з розроблення програмних систем. Необхідність співпраці кластерів безпосередньо встановлюється MSF-методологією [61, 65].

Як протилежність методології водоспаду ми можемо розглядати гнучкі методології. До них належать екстремальне програмування та SCRUM. Agile-маніфест стверджує, що це є необхідною умовою для ділових людей та розробників, щоб вони працювали разом протягом усього проєкту [68, 69, 119, 125]. Він приділяє більше уваги довірі окремих осіб і взаємодії процесів та інструментів. У гнучкій методології важлива роль самоорганізації команд. У ній також говориться, що найбільш ефективний і дієвий спосіб передачі інформації при розробці комп'ютерних програмних систем та у процесі спілкування особисто у команді [72]. Передбачається, що через регулярні проміжки часу команда повинна подумати про те, як стати більш ефективною і скорегувати свою поведінку відповідно до цієї метою [195]. Роль групової динаміки у гнучких методологіях надзвичайно важлива. Взагалі взаємодія між людьми у межах організацій та їхні злагоджені дії зазначаються у будь-якій із розглянутих вище методологій.

Структура команд розробників комп'ютерної програмної значно впливає на процеси групової динаміки, а водночас і на успішність виконання поставленого завдання. Можна виділити три основних типи ролей у розробці комп'ютерної програмної системи:

– цілеспрямовані, самоорієнтовані та орієнтовані на взаємодію. Вони відзначаються лише мотивацією до роботи;

– цілеспрямовані і мотивовані до роботи, яку вони роблять, не самоорієнтовані і мотивовані особистим успіхом й визнанням;

– орієнтовані на взаємодію між членами розробників, а також мотивовані присутністю і дією співробітників.

Для належного підбору команди від менеджера вимагається знаходження балансу між членами різних їхніх типів у команді. Коректно сформований баланс інтересів фахівців у групі може допомогти побудувати згуртовану й успішну команду для вирішення найскладніших задач [127].

Оскільки команди з розробки комп'ютерних програмних систем охоплюють різних фахівців, зокрема проєкт-менеджерів, системних аналітиків, програмістів, архітекторів програмного забезпечення і систем, тестерів, фахівців із програмного забезпечення гарантії якості, група впровадження та перевірки, група керування конфігурацією, група програмного забезпечення та інтеграції, група технічної підт-

римки. Усі ці фахівці повинні мати певні знання, щоб якісно виконати свою роботу. Різні фактори впливають на вибір спеціаліста, якого буде залучено до команди розробників. Часто керівники команди беруть участь у тестуванні досвіду, знань конкретних мов, інструментів, технологій і платформ, компетенції, комунікативності та особистих рис [130, 132, 139, 143].

У команді для кожного із членів визначені певні ролі. Командними ролями можуть бути: генератор ідей (творча людина, здатна вирішувати складні завдання), ресурсний слідчий (найкращий кандидат для переговорів), координатор (утверджує цілі і керує їхнім досягненням), мотиватор чи формувальник (формує виклики і мотивує команду до дії), аналітик (дає ґрунтовну оцінку ідеям), натхненник команди (робить команду ще більш згуртованою), виконавець (часто ефективний, добре організований, перетворює ідеї у реальні справи), контролер (покращує процеси) і фахівець (високопрофесійний працівник) [148, 155, 157].

Ці ролі у більшій або меншій є загальними для розробки будь-якої комп'ютерної програмної системи. Їхня участь і вплив на командну роботу також повинні бути збалансованими. Дії спілкування та взаємодії, ймовірно для підтримки координаторів та мотиваторів, оскільки їхні психологічні типи схильні до організації групової динаміки. Люди, які беруть участь у розробці, впровадженні та обслуговуванні програмних систем, високо цінують розумних професіоналів. З обумовлених причин вони повинні взаємодіяти і працювати у групі або команді, щоб гарантовано досягти поставленої мети. Тому групова динаміка у розробці програмних систем вимагає особливої уваги та оперативного корегування для успішного завершення конкретної задачі [182]. Зокрема, програм-менеджери групи управління потребують обговорень та певних міркувань щодо розміру групи, структури її складу і методології розробки програмного проекту, яку закладено в основу реалізації комп'ютерної програмної системи.

2.3.2 Аналіз особистостей членів команд засобами моделі Енеограми

Енеограма особистості *(з грец. enneа – «дев'ять» і grammos – «фігура»)* – концепція типів особистості та взаємовідносин між ними. Вона описує дев'ять «глибинних підсвідомих (драйвів)» та їхній вплив на формування поглядів, емоційні і поведінкові стратегії

дев’яти типів людей – так званих енеамітів. Ця модель була розроблена у 70-х роках 20 століття й у своїй основі ґрунтується на працях Оскара Ічазо й Клаудіо Наранхо. Певний вплив на розвиток моделі також мали ідеї Георгія Гурджиєва.

Енеограма описує дев’ять типів особистості, які розміщені на колі з однаковими віддалями одна від одної. Таке відображення символізує рівноцінний внесок кожного типу особистості у створенні світу. Типи з’єднані між собою не тільки колом, а й стрілками, які утворюють складну внутрішню фігуру, що вписана в коло.

Стрілки символізують зміни у поведінці людей у стані стресу, а також у стані психологічного комфорту. Проте не зважаючи на дослідження прихильників Енеограм та деякі дані щодо їх застосування у психотерапевтичній практиці, ця модель ще не достатньо вивчена в академічних колах тому, що немає відповідних тестів щодо коректного визначення типів, а також в уніфікації змісту кожного енеатипу у різних наукових школах.

Зараз дослідники висвітлюють Енеограми, як метод самопізнання та розвитку. Модель енеаграми деколи критикують за допущення суб'єктивних інтерпретацій, що значно ускладнює її наукову перевірку та валідацію. Ця модель являє собою коло, у яке вписаний трикутник, що з’єднує 3–6–9 сектори та символізує «закон трьох», а також шестикутна «зірка», яка з’єднує сектори 1–4–2–8–5–7 і символізує «закон семи». Витоки та історія розвитку Енеограми як відображення топології особистості і сьогодні викликає жваві наукові дискусії.

Вважається, що першим ознайомив Західний світ із фігурою Енеограми Георгій Гурджньов. Основоположником та розробником сучасної Енеограми типів особистості вважається уродженець Болівії Оскар Ічазо. Його «Енегон Его-фіксації» разом із деякими іншими аспектами особистості, розташованими на колі фігури Енеограми, послужили основою для подальшого розвитку цієї моделі. Ічазо почав викладати власну програму з особистісного розвитку в 50-ті роки 20 століття. Його підхід, названий «Протоаналіз», поєднує використання Енеограми з іншими символами та ідеями. Оскар Ічазо заснував Інститут Аріка, який спочатку працював у Чилі, а згодом переїхав у США. Саме Ічазо є автором терміна «Енеограма особистості».

Продовжувачем ідей О. Ічазо визначення та деталізації психотипів особистості став Клаудіо Наранхо, психіатр родом із Чилі, який вперше познайомився з Енеограмами особистості на курсі Оскара Ічазо в Аріка. Він продовжив розвивати вчення Енеограм і з початку

70-х років почав поширювати у США своє розуміння Енеограм особистості. Ідеї Наранхо мали значний вплив на подальший розвиток вивчення Енеограм. Деякий час вивчення, які пропагував Наранхо, залишалося закритим, проте надалі воно стало віддаленим загалу. Кілька учнів Наранхо порушили контракт, що забороняв їм викладати Енеограми, і почали проводити свої навчальні семінари по Енеограмі.

Однією з перших відкриті семінари, присвячені Енеограм проводити Хелен Палмер, яка згодом написала ряд книг, присвячених застосуванню енеограм для самоаналізу, особистісного розвитку і вибудовування функціональних сімейних та робочих стосунків. Також енеограми почали викладати деякі єзуїтські-священики, розглядаючи це вчення крізь призму християнської духовності, але згодом воно було визнане офіційним Ватиканом як невідповідне духовному розвитку.

Так вчення про Енеограми почало швидко поширюватися спочатку у психотерапевтичних і духовних колах, а потім поступово проникаючи і в інші сфери застосування. Сьогодні не існує єдиного уніфікованого розуміння енеограми і можливостей застосування цього знання. Ічазо не визнає розробок Наранхо, Палмер і єзуїтів, називаючи їх неправильними інтерпретаціями Енеограми. Розуміння енеограми учнями і послідовниками Наранхо так само може відрізнятися у деяких деталях.

Сьогодні багато авторів продовжують проводити дослідження Енеограми і сфер її застосування. Серед відомих вчених варто перерахувати такі імена: Клаудіо Наранхо, Дон Річард Різо і Рас Хадсон, А. Х. Алмаас, Річард Рор, Елізабет Вейгл, Джинджер Лапід-Богда, Хелена Макані.

Більшість фахівців, які досліджують Енеограми особистості, стверджують, що у людей одного і того ж енеотипу риси характеру можуть дещо відрізнятися, залежно від того, який вплив чинять на них два інших сусідніх типи. Їх часто називають «крилами». Так, людина третього типу може мати «крила» у другий і четвертий тип. Те, що тип Енеограми розташований на колі, може вказувати на їх швидше континуальну, а не дискретну природу взаємодії. Отже, вивчення про Енеограми вважається, що у людини є базовий тип і може бути одне або два крила, які впливають на деякі риси її характеру, проте вони не змінюють її базового типу, і відповідно, її глибинної мотивації.

Стрілки, якими пов'язані типи всередині кола, дають більш повне розуміння динаміки характеру кожного типу. Деякі автори називають їх переходами в «стрес і спокій», деякі – напрямами «інтеграції та дезінтеграції». Загалом ці стрілки описують, як змінюються наші реакції і стратегії поведінки, коли ми опиняємось у комфортній, безпечній ситуації або, навпаки, в стані тиску і стресу. Отже, щонайменший четвертий тип значно впливає на базовий тип і мотивацію людини: два «крила» і дві «стрілки».

У кожного типу особистості виділяється три підтипи, або інстинктивних варіанти. Вважають, що підтип людини залежить від того, який інстинктивний драйв домінує у життєвих стратегіях людини. В Енеограмі виділяють три основні інстинктивні енергії: інстинкт самозбереження, сексуальний (його також називають близькістю або «1: 1») і соціальний. Залежно від провідного інстинкту, життєвий фокус людини спрямований на задоволення:

– потреб, пов'язаних із безпекою, здоров'ям і добробутом (самозбереження);

– потреб, пов'язаних із близькими стосунками з важливими людьми;

– потреб, пов'язаних із визнанням, статусом, становищем у суспільстві і належністю до груп (соціальний).

З огляду на теорію провідного інстинкту, в Енеограмі виділяється 27 підтипів, оскільки представники одного і того ж типу з різними провідними інстинктами будуть значно відрізнятися один від одного, у них будуть різні основні цінності і життєві стратегії. Відповідно до цієї теорії, провідний інстинкт – це та сфера життя, в якій найбільш яскраво проявляється глибинна мотивація типу. Наприклад, якщо глибинна мотивація 3-го типу пов'язана з прагненням до успіху і захоплення, то 3-ка з інстинктом самозбереження буде реалізовувати цю потребу за завдяки блискучим матеріальним здобуткам і гарному доходу, 3-ка з сексуальним інстинктом буде шукати захоплення партнером, прагнучи відповідати його або її очікуванням, а 3-ка з соціальним інстинктом буде прагнути отримати визнання в суспільстві (звання, титули, почесне становище в компанії або професійному середовищі).

Вважають, що у кожної людини є всі три інстинктивних драйви, однак один із них, як правило, домінує. Деякі фахівці в сфері досліджень Енеограм вважають, що другий інстинкт також може бути до-

бре розвиненим, проте третій інстинкт часто є значно менш розвиненим або мало виявленим у житті людини.

Однією із значних відмінностей вчення Енеограми від інших топологій особистості є концепція рівнів розвитку. Вважається, що характер і життєві стратегії людини того чи іншого типу не є чимось статичним. Більш того, рівень психологічного, соціального і міжособистісного функціонування людини не залежить від типу його особистості. Будучи представником одного і того ж типу, людина може бути задоволеною завдяки (щасливою) збалансованому життю, або може перебувати на досить низькому рівні психологічного і соціального здоров'я. У всіх типів енеаграм виділяються так звані «рівні розвитку», що описують рівень особистісного і міжособистісного функціонування людини. Всього виділяється 9 рівнів розвитку, які об'єднуються в 3 великі групи: здорові, середні і нездорові рівні розвитку. Розробниками цієї концепції є Д. Р. Різо і Р. Хадсон. У своїх працях вони описують основні життєві стратегії кожного типу на кожному з дев'яти рівнів, виділяють фактори, що впливають на перехід вниз або вгору за рівнями.

Здорові рівні – це рівні високого функціонування особистості. На них людина має високий ступінь усвідомленості, балансу і свободи у виборі життєвих стратегій. На здорових рівнях людина стає більш вільною від патернів (шаблонів) свого типу. У людини є великий доступ до її сутнісного стану, і вона успішно реалізує свої особливі таланти та якості в своєму житті, а також професійній діяльності. Важливі життєві потреби задовольняються з легкістю.

На середніх рівнях, за позиціями авторів цієї концепції, перебуває більшість людей. На цих рівнях у центрі життєвого фокуса людини є задоволення важливих для неї потреб (визнання, безпеки, поваги тощо). На середніх рівнях людям вдається реалізовувати їх потреби, однак найчастіше це відбувається завдяки соціальним ролям, боротьбі, конфліктам, неусвідомленому маніпулюванню. На цьому рівні люди можуть розуміти підсвідомі мотиви і відстежувати патерни свого типу, проте їм складно протистояти різним викликам. Спочатку виникає реакція, а потім усвідомлення.

Багато науковців, які досліджують Енеограми вважають цю частину вчення особливо цінною, а також дає змогу визначити поточний рівень функціонування людини, зону найближчого розвитку та дестабілізації чинників. Графічне зображення моделі Енеограми особливостей психотипів членів команд розробників програмних систем із

можливими відхиленнями від базових типів у вигляді «крил» подано на рис. 2.4.

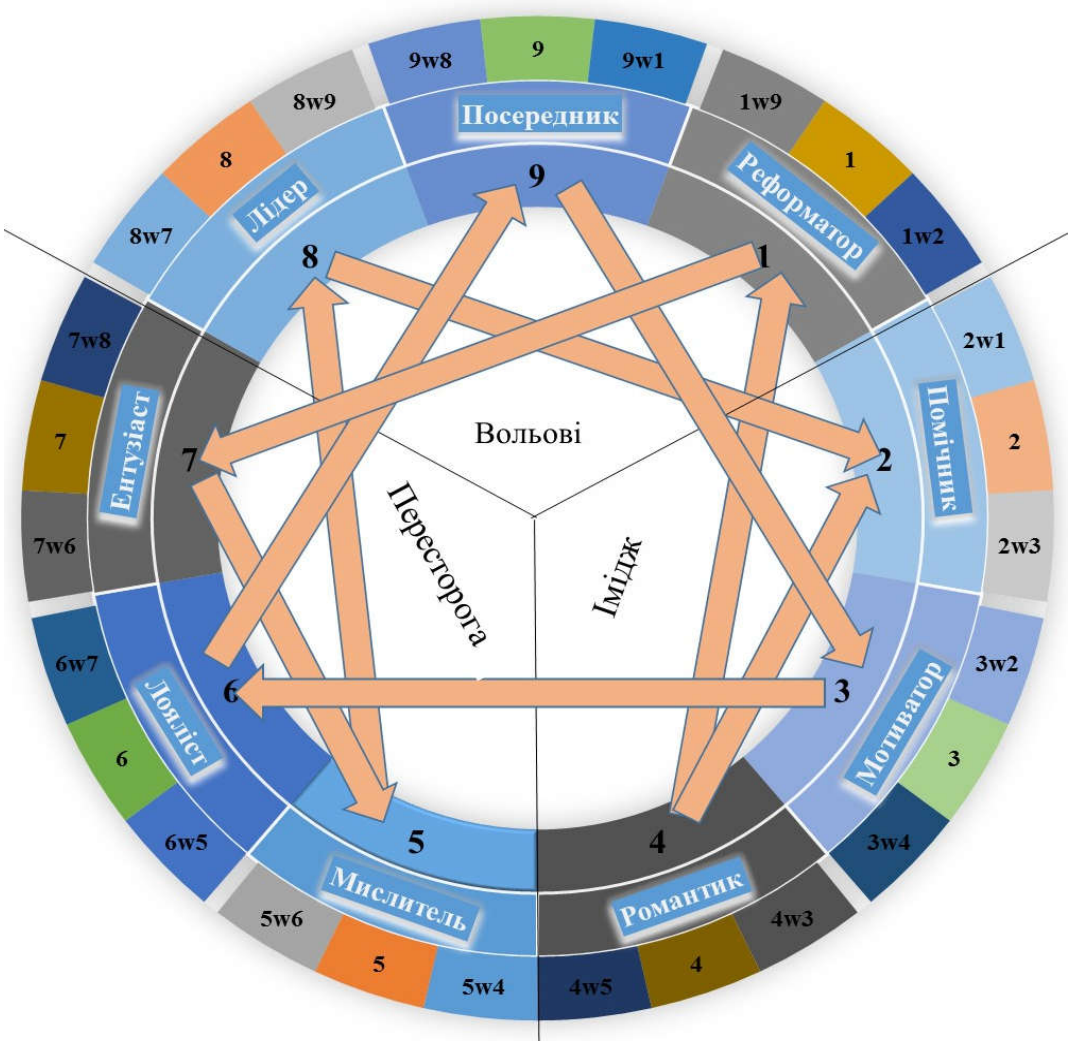


Рисунок 2.4 – Модель Енеограми особистих психотипів членів команд розробників та їх взаємозв'язків

Основні типи особистих рис із можливими «крилами» Енеограми описані у табл. 2.2.

Таблиця 2.2 – Базові типи особистих рис членів команд

Психо-тип	Крила	Назва психотипу англійська	Назва психотипу українська
1	**2**	**3**	**4**
1	1w9	the idealist	ідеаліст
	1	the reformer	реформатор
	1w2	the advocate	адвокат
2	2w1	the servant	слуга
	2	the helper	помічник
	2w3	the host, hostess	господар, господиня

Продовження табл. 2.2

1	2	3	4
3	3w2	the charmer	чарівник
	3	the achiever	мотиватор
	3w4	the professional	професіонал
4	4w3	the aristocrat	аристократ
	4	the individualist	романтик
	4w5	the bohemian	богемний
5	5w4	the iconoclast	іконоборці
	5	the investiguator	мислитель
	5w6	the problem solver	вирішувач проблем
6	6w5	the defender	захисник
	6	the loyalist	лояліст
	6w7	the buddy	приятель
7	7w6	the intertainer	співрозмовник
	7	the enthusiast	ентузіаст
	7w8	the realist	реаліст
8	8w7	the maverick	диспетчер
	8	the challenger	контролер
	8w9	the bear	ведмідь
9	9w8	the referee	рефері
	9	the peacemaker	посередник
	9w1	the dreamer	мрійник

Ґрунтуючись на дослідження в сфері формування робочих груп із використанням визначених особистісних психотипів членів можна сформувати команди розробників програмних систем для проведення експерименту щодо доцільності застосування цих методів в ІТ-галузі.

2.3.3. Чинники, що впливають на групову динаміку команд розробників

Успішне завершення будь-якої розробки комп'ютерної програмної системи залежить від різних факторів. З погляду групової динаміки можна виділити такі фактори: люди в групі, група в організації, технічні та управлінські комунікації.

Головне у враховуванні складу команди – це правильний баланс різноманітних навичок, досвіду та особистих рис. Група в організації

допускає, що робота повинна бути організована найбільш ефективно у такий спосіб, щоб кожен член команди міг сприяти досягненню поставлених завдань та цілей. Гарне спілкування між членами команди та зовнішні зацікавлені сторони і фактори повинні підтримувати виконання проєкту.

Часто групі зазвичай не вистачає їх потенціалу з погляду продуктивності. Групова продуктивність зменшується внаслідок таких проблем, як погане управління ресурсами та соціальне плавання, де окремі члени групи не приділяють свою увагу групі. Отже, методи групової динаміки у розробці програмних систем (проєктів) впливають такі п'ять факторів:

1. Розмір групи. На практиці розмір групи містить від 5 до 12 осіб. Оптимальна кількість членів – приблизно 10 осіб. Це число підтримує створення всього необхідного, у тому ж числі і комунікаційні зв'язки всередині групи.

2. Структура групи. Ієрархічна структура груп має деякі переваги та недоліки, які необхідно враховувати при управлінні групою. Наприклад, члени одного й того ж рівня можуть мати меншу взаємодію між собою у випадку жорсткої організації комунікаційних потоків та взаємної конкуренції.

3. Структура груп. Вік, стать, національність та психологічні ролі, вказані вище повинні бути належно змішані, щоб забезпечити ефективну роботу у команді розробників.

4. Фізична робоча обстановка. Організація робочого місця може заохотити членів команди до комунікації. Є працівники, що віддають перевагу окремим місцям, а інші люблять відкритий простір.

5. Доступні канали зв'язку. Сьогодні спілкування віч-на-віч є не єдиним можливим способом поділитись ідеями та результатами. Інтернет надає широкі можливості для організації віртуальних і мобільних команд, онлайн – конференцій та зустрічей, об'єднуючи учасників із різних куточків світу.

Проблеми організації колективної роботи та групової динаміки є надзвичайно важливими поточними умовами у досліджених компаніях з розробки програмних систем. Цінний вплив комунікації та взаємодії у командах розробки програмних систем детально у Моделі COCOMO II [18].

За моделлю ці фактори впливають на працю, інтенсивність, організаційні характеристики команди розробників комп’ютерних програмних систем, зокрема послідовні цілі, психологічна сумісність фахівців і, досвід роботи членів у команді та інші об'єктивні та суб'єктивні особливості зацікавлених учасників сторін проєкту [166]. Особис-

та мотивація психологічні особливості та довіра також відіграють велику роль в його успішному виконанні. Усі ці характеристики можуть бути інтегровані в якісний показник, який впливає на інтенсивність праці, значення коефіцієнта командного з'єднання, який відповідає певному рівню інтенсивної співпраці. Коефіцієнт згуртованості команди використовується для оцінки інтегральної взаємодії, яка впливає на вартість розробки програмної системи разом із такими перевагами, як гнучкість розробки, розподіл ризиків та зрілість процесу (табл. 2.3).

Таблиця 2.3 – Вплив згуртованості команди на трудомісткість завдань

Характеристика завдання	**Значення трудомісткості завдання**				
	дуже низьке	**низьке**	**оптимальне**	**велике**	**дуже велике**
Узгодження цілей	Мінімальне	Незначне	Відносне	Велика	Всього
Можливість переривання з іншими членами команди	Низька	Незначна	Відносна	Великий	Всього
Досвід роботи у цій команді	Відсутній	Невеликий	Достатній	Великий	Дуже великий
Ступінь довіри і взаємодії у колективі	Відсутній	Достатньо низький	Певною мірою	Великий	Дуже великий
Узагальнений рівень колективної роботи	Мала взаємодія	Ускладнена взаємодія	Колективна робота	Високий ступінь взаємодії	Постійна взаємодія
Цінність згуртованості команди	5,63%	4,57%	3,47%	2,26 %	1,28 %

Згуртованість команди є визначальним фактором масштабу проєкту із розробки комп’ютерних програмних систем, який впливає на

їхню вартість. Методи групової динаміки мають прямий вплив на згуртованість команди, що своєю чергою, впливає на проєктні витрати. Отже, ідентифікація та оцінка групової динаміки впливає на технічні наслідки та водночас є складним завданням.

2.3.4. Гендерний підхід та національне різноманіття при формуванні команд

Організація розробки комп'ютерних програмних систем – це виробнича діяльність, організована навколо командних середовищ. Реалізація командних структур не є простим процесом і не обов'язково призводить до успіху, не достатньо лише об'єднати людей у колектив і припустити, що всі погоджуються з тим, що потрібно робити (Allen and Hecht, 2004). У такому контексті будь-який конфлікт може вплинути на продуктивність команди. Керівники груп зацікавлені у тому, щоб запобігти конфліктам, уникнути їх або, в найгіршому випадку, керувати конфліктами, що можуть виникнути, та повинні розуміти ключові фактори, які роблять команду злагодженою й продуктивною щодо поставлених замовником завдань і цілей.

Розмаїття у робочих групах вивчалось у декількох дослідженнях (Horwitz and Horwitz, 2007; Stahl et al, 2010) і розглядається як будь-який атрибут, який відрізняє людей (Williams and O'Reilly, 1998), наприклад, демографічні ознаки (вік, стать, національність), функціональні (наприклад, роль, володіння, експертиза) або суб'єктивні особливості (наприклад, особистість) [115].

У попередніх дослідженнях вчені відображали протилежні думки про роль різноманітності у командній роботі. Деякі дослідники відображали суттєві позитивні кореляції між різноманітністю та продуктивністю (Earley, Mosakowski, 2000), тоді як інші вчені стверджують, що різноманітність негативно впливає на результати роботи команди (Watson et al, 1993) [13].

Позитивний результат дослідження було підтверджено по семи із вісьмох гіпотез, а саме расова різноманітність команди пов'язана зі збільшенням доходу від продажів, більшою кількістю клієнтів, більшою часткою ринку та більшим відносним прибутком. Гендерна різноманітність команди була пов'язана зі збільшенням доходу від продажів, більшою кількістю клієнтів та більшим відносним прибутком. Щодо групової розробки програмних систем із відкритим вихідним кодом, то Daniel та інші вчені (2013) вивчали вплив різноманітності

на залучення громад та ринковий успіх у вибірці з 417 проєктів SourceForge. Результати показали, що репутація та різноманітність ролей позитивно співвідносяться з успіхом на ринку та міждержавним залученням членів команд. І навпаки, різноманітність розмовних мов та національностей негативно пов'язується із залученням членів команди та позитивно впливають на успіх ринку.

Дослідження впливу різноманітності програмних груп на обсяг виконаної роботи та на поведінку членів у середині групи виявили, що збільшення різноманітності досвіду роботи збільшило продуктивність групи та зменшило міграцію членів проектної групи до граничного значення. Проте різноманітність у широкому розумінні цього терміна – це позитивний фактор, що приносить користь проєктній групі розробників програмних систем, які працюють разом. Різноманітність у груповій динаміці стимулює позитивні зміни. Наприклад, люди з різним культурним походженням повинні розуміти, як спілкуватися і взаємодіяти, оскільки концепція «гарних звичаїв» в іншій країні може відрізнятися від тієї ж концепції у власній країні.

У нашому дослідженні ми акцентуємо увагу на вплив гендерного та національного різноманіття на продуктивність команди з розробки програмних систем. На основі системного аналізу наукових досліджень ми використовували питання фіксації часу і ввічливості як проксіметрики для продуктивності команди та рівня зв'язку відповідно. Отже, зосередивши нашу увагу на розумінні впливу гендерних або національних чинників, ми з’ясували, що фактор національності може бути репрезентативним для широкого культурного фону (наприклад, мови), інакше важко його обчислити.

Розглянемо такі чинники у нашому дослідженні:

– чи пов'язане гендерне або національне розмаїття з питанням, що визначає час роботи команди;

– чи гендерна різноманітність у групах була пов'язана з меншим часом фіксації проблем;

– чи пов'язане гендерне або національне різноманіття з загальною ввічливістю команди;

– чи пов'язане гендерно-національне різноманіття з загальним настроєм команди?

Отже, запропонований нами науковий підхід показав, що гендерна різноманітність є провідною метрикою, який водночас пояснює дисперсію даних, а також позитивно пов'язаний із продуктивністю у командах.

Гендерна різноманітність у командах розробників програмних систем була пов'язана з більш позитивними настроями. Наш метод формування команд розробників програмних систем засвідчує, що гендерна різноманітність у командах була провідною метрикою, що пояснювала дисперсію даних та мала позитивний вплив на настрої команди (вона мала тенденцію до підвищення ввічливості).

Відповідно до результатів системного аналізу дослідження можна стверджувати, що є достатньо багато команд, які мають гендерну різноманітність, незважаючи на те, що жінки мають «проти стереотипні поведінки» однак чоловіки і жінки, які пов'язані з обчислювальною технікою переважно внаслідок культурних і екологічних умов, підкреслюють той факт, що гендерна нерівність є культурним наслідком.

Для того, щоб організація у команді розробників програмних систем була ефективною, необхідно проаналізувати, як люди працюють разом, а також те, що вони роблять. Традиційно команди розробників будуються в ієрархій із менеджером у верхній частині. За допомогою новітніх інноваційних технологій (таких як системи відстеження проблем, гнучкі методології розробки, інформаційні дошки тощо) ми спостерігаємо зміну парадигми у тому, як люди само організовують свою проектну роботу.

Навіть розробники програмних систем, особливо у середовищах з відкритим кодом, починають самоорганізовуватися за межами ієрархії, працюючи більш відповідально у спільних підходах, застосовуючи водночас більш плоскі структури. Зсув в ієрархії розглядає колекції (команд) людей, подібних до живих систем, де необхідно аналізувати всі окремі частини і стосунки разом. Отже, жінки частіше звертаються із запитами, ніж чоловіки, проте рівень сприйняття жінок вищий лише тоді, коли вони не ідентифіковані як жінки. У контексті теорії гендерної проблематики на робочих місцях розробників програмних систем правдоподібні пояснення включають наявність ґендерної упередженості у відкритому коді, упередженості та самовибору, а жінок – до більш високих стандартів роботи.

Щодо розробки програмних систем, то адекватний вибір відповідних пристроїв зв'язку є важливим питанням, оскільки він може суттєво вплинути на результат комунікації. Отже, більш високий рівень комунікації позитивно співвідноситься з меншим часом фіксації, водночас ми можемо припустити, що комунікація позитивно пов'язана з більш високою продуктивністю праці. Крім того, коректна комунікативна поведінка у команді із розробки програмних систем виявилася

більш привабливою для замовників. На підставі цих тверджень доцільно припустити, що комунікативність членів команд можна було б використати як основу для оцінки якості та рівня спілкування у групі. Тому ми вирішили розглядати комунікативність як одну із залежних змінних нашого дослідження. Для прикладу розглянемо модифікований евристичний метод, який оснований на оптимізації модульності підходів при формуванні команд розробників програмних систем (рис. 2.5).

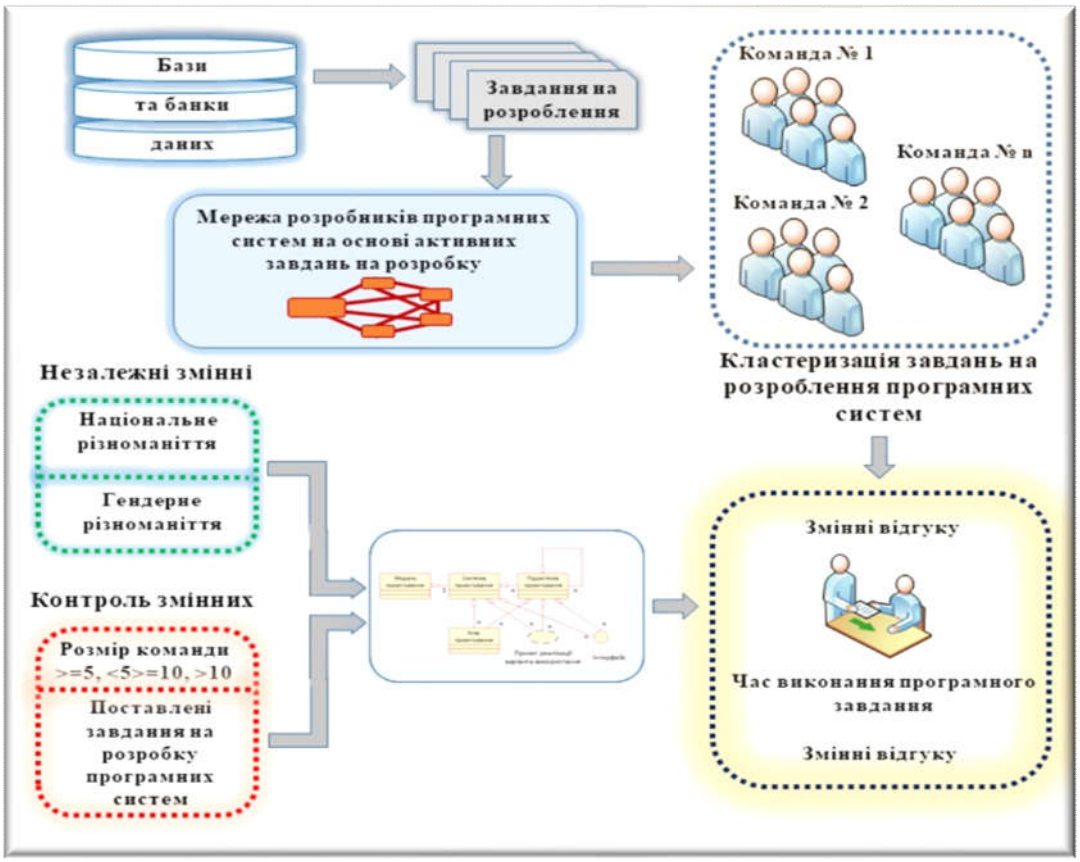


Рисунок 2.5 – Модель формування команд на основі національних та гендерних принципів

У цій моделі представлено розповсюдження набору даних, щоб визначити залежність гендерності та країни походження розробників програмних систем, акцентовано питання співпраці і проведено аналіз модульності для отримання ефективних команд. Продуктивність команди розробників програмних систем припускає безперервний діапазон значень, але його статистичний розподіл – це значення, що можна спостерігати через значну величину порядків, більшу за інші значення [171-173]. Для таких розподілів важко вибрати теоретичну

криву, яка найкраще підходить для емпіричних даних між різними можливостями через весь діапазон значень.

Отже, у дослідженні ми представили аналіз зв'язків між різноманітністю команди з погляду національності та гендеру, загальної ввічливості, настроїв та продуктивності команди. Ввічливість – це фактор, який, безумовно, допомагає зменшувати конфлікт і ситуацію між людьми, а також настроями, тому сприяє висвітленню важливості та впливу двох факторів, які необхідно враховувати при побудові команди: гендерне та національне різноманіття.

Розробка програмних систем командами із різних континентів лише зростатиме. Отже, розуміння різних культур і звичаїв та толерантне ставлення до процесу розробки програмних систем ставатимуть все істотнішими. На нашу думку, це дослідження є оптимальною відхідною точкою для стимулювання подальшої наукової діяльності у цьому напрямі і надалі може допомогти керівникам команд у прийнятті найефективніших рішень під час вирішального етапу створення команди розробників.

2.4. Адаптивне тестування знань претендентів для підбору у команди розробників програмних систем

Як відзначає М. Б. Челишкова, в останні роки в практиці перевірки набутих фахових компетенцій складається ситуація, під впливом якої відбувається традиційне тестування за допомогою стандартних тестів фіксованої довжини, модифікується в сучасні ефективні форми адаптивного тестування. Ці тести ґрунтуються на математичних моделях адаптивного тестування, які відрізняються від традиційних теоретико-методологічних основ та інших технологій конструювання й представлення тестів.

У продовж декількох останніх років питання адаптивного тестування неодноразово розглядалося зарубіжними ученими. Про це свідчать дослідження фундаментального характеру і публікації таких авторів, як J. Arter, R. Hambleton, J. Horn, C. Jensema, F. M. Lord, J. Millman, L. Nauels, R. Owen, K. Patience, M. Reckase, J. Spray, H. Swaminathan, V. Urry, M. Waters, R. Wood, A. Zara та багатьох інших.

З початку 90-х років комп’ютерне адаптивне тестування набуло за кордоном широкого визнання у сфері практичної реалізації. У цьому суттєво сприяли прикладні теоретичні дослідження таких вчених,

як C. Bunderson, D. Inouye, G. Kingsbury, J. Olsen, H. Wainer, D. Weiss, праці яких були підтримані не тільки педагогічною спільнотою, але і знайшли широке використання в практиці професійного відбору у структурах промислового та військового комплексів, а також галузі інформаційної технології.

Стрімкий розвиток електронної перевірки фахових знань ґрунтується на значних перевагах, одна з яких – це можливість індивідуального підходу до тестування рівня знань та вимог в отриманні нових навичок. Якщо система комп'ютерного тестування здатна автоматично оцінити рівень знань та можливостей претендента, то електронний тест може бути адаптований під кожного претендента окремо. Надзвичайно важливою задачею такого підходу є розробка методів адаптивного тестування при підборі кадрів у групи розробників програмних систем.

Сучасне тестування – це комплекс стандартизованих методів вимірювання латентних (тобто недоступних для безпосереднього спостереження) параметрів людини, які визначають рівень її підготовки і відповідність фаховим стандартам у когнітивній області знань. Водночас широко використовуються математичні методи планування й обробки результатів тестування, а також сучасні технології обробки інформації.

Об'єктивний контроль знань, вмінь і навичок – одне з актуальних завдань нашого часу. Його вдається виконати за критеріально-орієнтованої інтерпретації тестування. Критеріально-орієнтоване тестування призначене не тільки для оцінювання рівня компетенцій, а й для визначення рівня індивідуальних досягнень щодо певного критерію на підставі логіко-функціонального аналізу завдань.

Важливою задачею є вибір технології й архітектурного підходу, який дав би змогу створити гнучку платформу для побудови систем тестування, а також легкого її налаштування на будь-яку предметну область будь-якої тестової платформи без обмеження, на види тестування чи на типи тестових питань або форматів тестів. Така платформа повинна мати чітко визначений опис інтерфейсів і основних компонентів системи для подальшої успішної тісної інтеграції з будь-якими системами перевірки фахових знань без потреби вносити зміни у вихідні коди системи тестування.

Окрім цього, важливою також є можливість автоматизації побудови тестових завдань з орієнтацією на побудову валідних, надійних і функціонально повних тестів. Для визначення цих характеристик за-

стосовується статистична обробка матеріалів тестового контролю. Аналіз завдань тестування математичними методами дає змогу отримати інформацію про їхні приховані дефекти, що не можуть бути виявлені за допомогою експертних методів.

2.4.1. Математична модель тестування фахових знань претендентів у команду розробників програмних систем

У теорії просторів знань значна область знань може бути подана як множина елементарних знань та навиків, які може опанувати претендент у цій предметній області, де множина знань це $M = \{m_1, m_2, m_3, ..., m_n\}$. Рівень (стан) знань претендента z у цій предметній області розглядається як підмножина множини знань $z \subseteq M$. Якщо елементарне знання претендента у групу розробників програмних систем $z_m \in M$ таке, що його наявність вимагає також елементарних знань $z_n \in M$, то вважають, що z_n є пре реквізитом для z_m. Отже, множина знань із заданими відношеннями між його елементами визначає множину компетенцій. Множина знань повністю визначає множину Z всіх можливих станів знань, у яких може перебувати претендент.

Тестування претендента у групу розробників програмних систем передбачає процес, у якому комп'ютерна тестова система вибирає із можливих станів знань Z – стан, за якого він відповідає дійсному рівню знань претендента. Застосовуючи адаптивні сценарії, комп'ютерна тестова система у процесі тестування поступово вилучає ті стани знань, які не досягають дійсного стану фахових знань, що ґрунтується на основі відповідей претендента на попередні тестові запитання. Комп'ютерна тестова система пропонує на наступному кроці тестові запитання лише по тій підмножині знань, відносно якої вона ще не може визначити рівень фахових компетенцій претендента. Отже, система сама адаптується до рівня фахових компетенцій, виключаючи із тесту питання, які будуть для претендента занадто складними або дуже простими. Коли досвідчений екзаменатор проводить усний іспит, він практично завжди застосовує певний спрощений інтуїтивний варіант адаптивного тестування. Після декількох вдалих відповідей претендента екзаменатор намагається ставити питання складніші, і коли претендент успішно справився із цим складнішим питанням, то екзаменатор справедливо заощаджуючи час, ставить

претенденту відповідну (валідну) оцінку. Після декількох неправильних відповідей екзаменатор, навпаки, знижує складність питань – ставить найпростіші питання і якщо відповідь негативна, екзаменатор ставить відповідну оцінку.

Отже, для визначення крайніх випадків, необхідно значно менше тестових завдань для забезпечення практично того ж рівня оцінки претендента у групу розробників комп'ютерних програмних систем. Значно більше часу та питань потрібно використати для середняків, які з одним завданням можуть впоратися, а з іншим ні, оскільки недостатньо добре опанували предметну область, яка винесена на тестування. Суть адаптивності алгоритму тестування полягає у тому, що тест самостійно пристосовується до наступного претендентові рівня складності, і тим самим швидше визначається фаховий рівень знань претендента за відповідною шкалою.

Метою проведення комп'ютеризованого адаптивного контролю фахових знань у цьому дослідженні є розробка математичної моделі та алгоритму поведінки системи адаптивного тестування. Ми наведемо результати аналізу математичної моделі адаптивного тестування, яка ґрунтується на ймовірнісному підході до оцінки рівня знань претендента у команди з розробки програмних систем.

Кожному можливому стану P_0 із множини Z ставиться у відповідність величина, яка характеризує апріорну оцінювальну ймовірність того, що претендент перебуває у цьому стані $P_0(z)$.

Після того, як претенденту поставили питання на тему m_i із множини фахових знань та оцінили правильність вірності відповіді, здійснюється переоцінка ймовірностей перебування претендента у станах знань із множини Z: $M_k(z_1), M_k(z_2) \ldots M_k(z_M)$.

Якщо відповідь на питання з теми m була правильною, то ймовірності станів, які мають у собі тему m_i, збільшуватись, а оцінка інших станів –зменшуватись. Водночас на кожному кроці тестування зберігається норма $\sum_{\forall z \in Z} M_k(z) = 1$.

У дослідженні ми подаємо подальший розвиток вивчення моделі, а саме аналіз збіжності процесу до стану значень, які шукаємо у випадковій послідовності тестових запитань.

Переваги адаптивності залежать не тільки від економії часу, а і від обмеженої кількості висвітлених питань з бази знань. Оскільки запитань одному претенденту у команду розробників комп'ютерних

програмних систем ставить менше із банку питань, то менш прозорим для претендентів стає повний банк тестових фахових завдань загалом. Використання тематичної блочної структури у межах однієї предметної області разом з використанням адаптивних алгоритмів тестування, не тільки заощадить комп'ютерний час, а й дасть змогу швидко та надійно локалізувати проблемні зони – провали у засвоєнні матеріалу претендентом.

Тести можуть розрізнятися за орієнтованістю, а також і за типом побудови тестових питань. За орієнтованістю розрізняють критеріально-орієнтований тест і нормативно-орієнтовний.

Критеріально-орієнтованим вважають тест, що призначений для виміру тієї частини фахових компетенцій, яку опанував претендент, або для визначення відповідності претендента у команду розробників програмних систем визначеному критерію. Обидва типи можуть містити однакові завдання, різниця полягає в опрацюванні та інтерпретації результатів. За типом генерації питань розрізняють два види тестів – тести, що є завжди не змінними (авторські), і тести, що генеруються за допомогою генератора випадкових чисел зі сховища бази питань.

Авторський тест – це набір чітко визначених питань, які постійні для кожного тесту. Кожен претендент отримує цей набір тестових питань для перевірки фахових компетенцій без змін. Такий тест доцільно використовувати у процесі засвоєння знань для самоконтролю та проміжного контролю претендентів на займання вакантних посад у групі з розробки комп'ютерних програмних систем.

Тест дає змогу побачити рівень успішності кожного із претендентів окремо та групи загалом й прогалини у тестових питаннях під час складання тестів. Комп'ютерні тести, що генеруються випадково – це динамічно змінні тести, при створенні яких визначається тематика цього тесту, а також кількість питань з кожної теми, час на проходження тесту, а у деяких випадках і оперативна зміна складності питань (адаптивні тести).

Тест генерується автоматично на основі статичних або адаптивних даних, вибірки зі сховища банку питань. Статичний комп'ютерний тест за умови великого банку даних питань(від 600 до 1.000 питань) гарантує генерацію практично повністю різних тестів, проте ускладнює формування питань та варіантів відповідей. Цей тип зручно застосовувати для контрольних тестів. Такий тест дає змогу автоматично створити індивідуальні тести для кожного претендента і

перевірити його власний рівень знань та набуті компетенції. Різновидом статичного тесту є адаптивний тест з адаптивним алгоритмом формування тестових питань, тобто тест, завдання якого подають претенденту залежно від того, як він виконав попереднє завдання або їхня сукупність [279].

Загальне правило адаптивного тесту – це те, що ефективне виконання декількох завдань певної складності є підставою для подання наступного пулу завдань складніших. Невдале вирішення пулу завдань – це підстава для подання менш складного завдання, тобто порядок питань залежить від попереднього пулу відповідей претендента у команду розробників програмних систем. Отже, адаптивний тест являє собою варіант автоматизованої системи тестування із заздалегідь відомими параметрами складності, що у водночас дає змогу розробникам бази знань тестових запитань зменшити кількість тестових питань до 300 з обов'язковою розбивкою питань за рівнями [305].

2.4.2. Багато рівнева модель адаптивної перевірки знань претендентів у команди розробників програмних систем

Модель адаптивного подання знань передбачає їхню структуризацію знань та її формалізацію. Проблеми проєктування адаптивних тестових завдань формують головне завдання під час розробки автоматизованих адаптивних тестових систем і розробки багаторівневої моделі контролю знань (рис. 2.6).

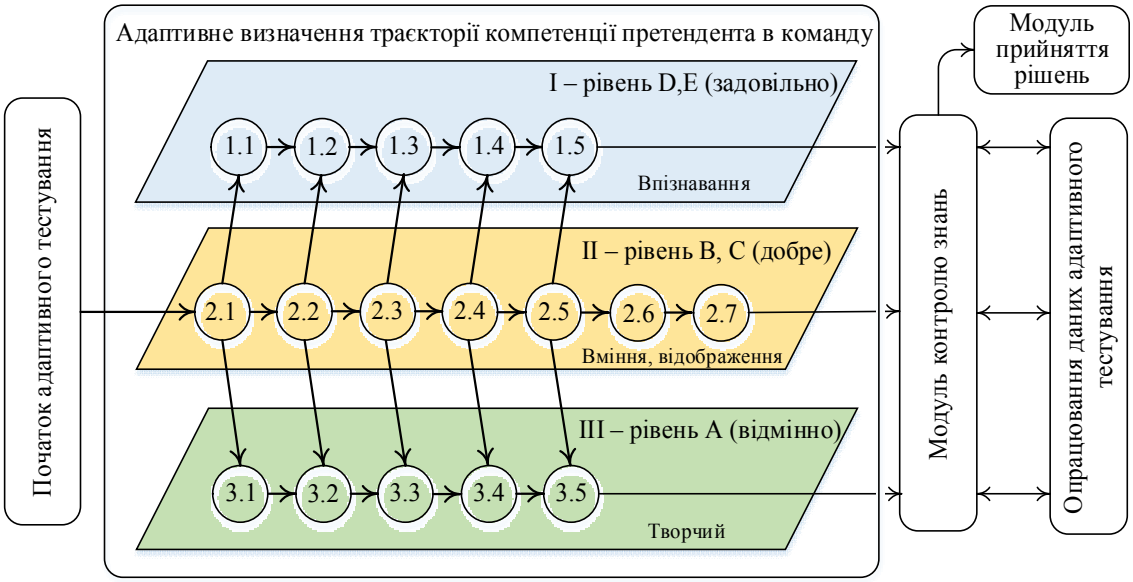


Рисунок 2.6 – Багаторівнева модель адаптивної перевірки знань претендента

Кількість рівнів деталізації складності ієрархічної моделі знань предметної області залежить від ступеня деталізації понять. Такий підхід до організації адаптивних тестових систем дає змогу значно скоротити час тестування, зменшити обсяг пам'яті, займаних базою знань та даних.

В якості адаптивної траєкторії компетенції претендента у групу розробників комп'ютерних програмних систем у когнітивній області пропонується: класифікація типів об'єктів (творчий, вміння, відображення, впізнання), виділення деяких фундаментальних видів зв'язків між об'єктами.

Прикладом використання ієрархічної структури у тестовій системі є INDIGO (Indigo Software Technologies), яка складається з питань і груп питань. Групи у водночас можуть мати як питання так і інші групи. Зараз в адаптивних тестових системах доступні п'ять типів питань: вибір одного варіанта відповіді, множинний вибір правильної відповіді, введення відповіді з клавіатури, установка відповідності, розстановка у потрібному порядку. Система опрацювання результатів оцінювання знань реалізує тестування претендентів у групи розробників програмних систем за індивідуальною траєкторією для визначення заданого рівня компетенції. Процес тестування реалізується відповідно до адаптивної моделі тестування. Індивідуальна траєкторія тестування містить тестові завдання 3-х рівнів складності: оцінка знань модуля на рівні «3», «4», «5». Ця шкала є універсальна. Так, її можна записати і у вигляді 100 бальної шкали оцінювання, розбиваючи результати на діапазони, які будуть визначати надані рівні складності.

Для кожного тестового завдання вводиться інтегрований коефіцієнт знання z, що враховує засвоєння поточного матеріалу і параметри як ступінь опанування матеріалу, ваговий коефіцієнт відповіді, спрямований на варіювання порогової величини рівня складності проходження завдання. Для успішного проходження певного рівня складності тестового завдання претендент у групу повинен отримати коефіцієнт $z > 0,7$. Отже, адаптивна система буде автоматично надавати різні рівні складності тестових завдань, надаючи реальні фахові компетенції претендента.

Розглянемо адаптивну модель тестування на основі автоматизованого підходу, задану кортежом:

$$A = \{X, Y, S, \delta, \lambda\},$$

де $X = \{X_0, X_1, X_2\}$ – вхідний рівень, який відображає значення результату рівня складності тестових завдань;

X_0 – результат тесту при значенні $z = 0,8$;

X_1 – результат тесту при значенні $z > 0,8$;

X_2 – результат тесту призначенні $z < 0,8$;

$Y = \{Y_0, Y_3, Y_4, Y_5\}$ – вихідний рівень, який відображає результат тесту (оцінка). Y_0 – оцінка «2»; Y_3 – оцінка «3»; Y_4 – оцінка «4»; Y_5 – оцінка «5»; $S = \{S_0, S_1, S_3, S_4, S_5\}$ – рівень внутрішніх станів, який відображає рівень складності тестових завдань і точку зупинки системи. S_1 – зупинка роботи системи, S_3 – складність питань на оцінку «3», S_4 – складність питань на оцінку «4», S_5 – складність питань на оцінку «5»; S_0 – початковий стан системи (у момент часу $t = 0$), визначається залежно від індивідуальної траєкторії навчання, групи питань для визначення наступного рівня складності тестових завдань;

$\delta(S_i, X_j)$ – функція переходів, здійснює, залежно від стану і вхідного значення рівня, перехід до наступного рівня складності тестових завдань або завершення тесту;

$\lambda(S_i, X_j)$ – функція залежності від стану і вхідного значення рівня оцінки, яку отримав претендент у команду розробників програмних систем. Модель опрацювання даних подана на рис. 2.7.

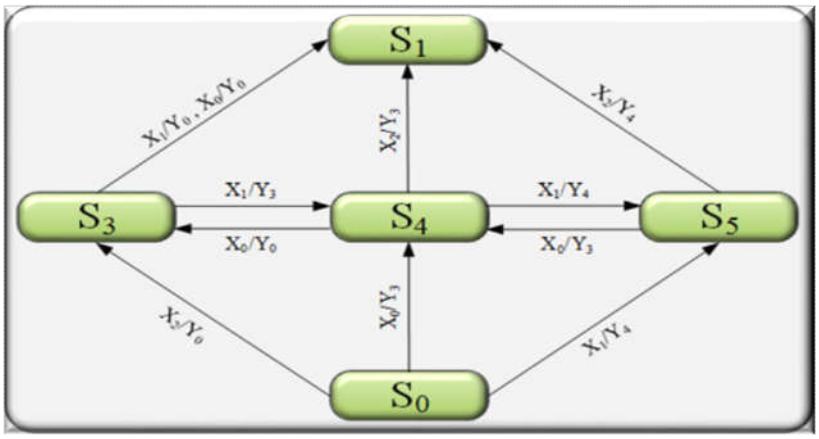


Рисунок 2.7 – Багато рівнева адаптивна модель опрацювання даних

Адаптивна система дозволяє виявляти ступінь засвоєння фахового матеріалу, видаючи на виході результат, який буде використовуватися для рішення про перехід до наступного етапу тестування.

Описана модель тестування дозволяє оптимізувати час навчання, так як стан S_0 тест орієнтований на інтегрований показник успішності і за нього він може отримати оцінки від двох до чотирьох. Таким чином, якщо у стані S_0 на вхід надходить результат X_1, то автоматизована система тестування переходить у стан S_5 і на виході з'явиться результат Y_4. У стані S_5 видаються тестові завдання вищої складності й оцінюються знання на оцінку «5». Система у цьому випадку прогнозує вихідний сигнал Y_4. Вихідний сигнал на переході у стан S_1 зберігається у динамічному профілі претендента у команду і може використовуватися як вхідний рівень для моделі переходів між модулями знань.

2.4.3. Автоматизована система оцінки знань претендентів для залучення у групи розробників програмних систем

Розглянувши запропоновані методи та алгоритми підрахунку підсумкової оцінки в адаптивних тестах, спроєктованої системи контролю компетенції претендентів, пропонуємо використати модель контролю знань у кілька кроків (рис. 2.8).

проходження основного тесту претенденти мають пройти тест вхідного контролю. Він містить питання з попередньої (суміжної) предметної області, яка є необхідним критерієм для успішного проходження поточного зрізу фахових компетенцій. Цей тест складається з *n*-ї кількості питань і слугує для оцінювання знань. Отже, на цьому кроці використаємо класичну формулу підрахунку балів за кожну правильну/неправильну відповідь (1):

$$X_i = \sum_{i}^{n} x_i \quad , \tag{2.1}$$

де x_i – бал за конкретне питання відповіді для X_i- того питання.

Підсумкову суму балів розраховуємо як суму балів за кожну отриману відповідь на X_i питання (2):

$$X = \sum_{i=1}^{n} x_i \, , \tag{2.2}$$

де X_i – сума балів за *n*-ну кількість питань тесту.

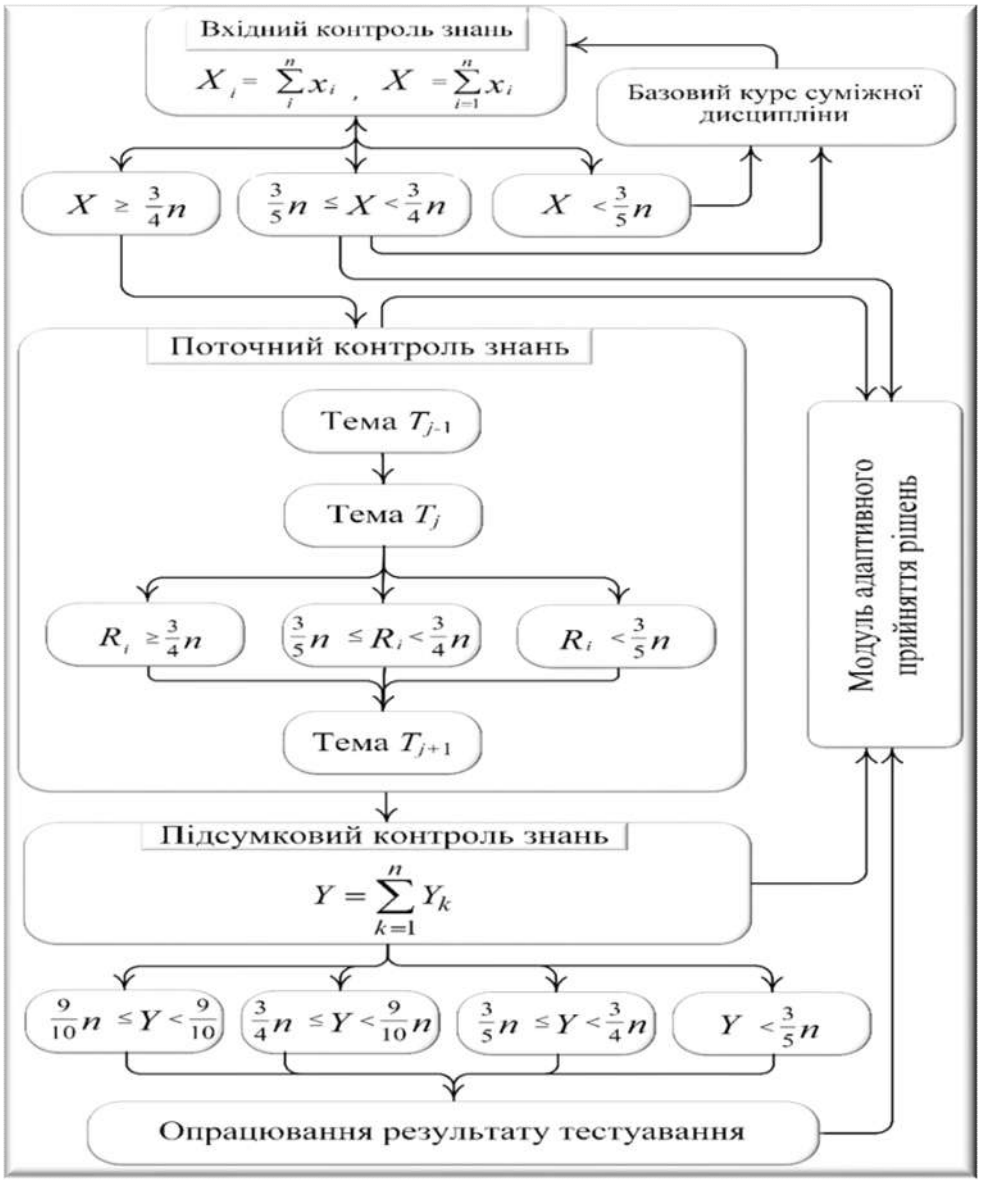


Рисунок 2.8 – Модель адаптивного навчання і тестування знань претендентів у групи розробників програмних систем

Перший крок – вхідний контроль фахових знань. Перед початком

З огляду на те, що знання претендента з суміжної області є необхідними для успішної здачі тесту із фахової компетенції, використаємо таку шкалу оцінювання знань:

- якщо $X \geq \frac{3}{4}n$, то претендент успішно пройшов вхідний контроль і має необхідні знання та фахові компетенції для тестування наступного рівня компетенцій, що перевіряються автоматизованим адаптивним тестом;

- якщо $\frac{3}{5}n \leq X < \frac{3}{4}n$, то у такому випадку пропонується ще раз пройти вхідний контроль знань, вказавши на допущені помилки. У

разі повторного отримання балів в межах $(\frac{3}{5}n; \frac{3}{4}n)$ претенденту рекомендують повторити базовий курс дисципліни з недостатнім рівнем знань та компетенцій самостійно;

- якщо $X < \frac{3}{5}n$, то базові знання претендента не достатні для успішного зарахування у групу розробників програмних систем. Претенденту рекомендують повторити базовий курс дисципліни з недостатнім рівнем знань та компетенцій самостійно;

Другий крок – поточний контроль знань. У разі успішного складання вхідного контролю знань претендент може проходити поточний контроль за темами, що винесені на тестування фахових компетенцій. Надалі формування інтегральної оцінки R_i з будь-якої теми тестової дисципліни –T_j будемо проводити згідно з формулою (2.2) із огляду на коефіцієнт складності питання:

- якщо $R_i \geq \frac{3}{4}n$, претендент успішно склав T_j тему та має право на складання наступної T_{j+1} теми адаптивного тесту;

- якщо $\frac{3}{5}n \leq R_i < \frac{3}{4}n$, претендент за допомогою модуля пояснень проходить цей тест у режимі навчання, а потім знову у режимі тестування, отримуючи питання I та II рівнів складності;

- якщо $R_i \geq \frac{3}{5}n$, претенденту пропонується повторне вивчення теми адаптивного тесту.

Третій крок – підсумковий контроль знань. Після закінчення кваліфікаційного адаптивного тесту на автоматизованій системі претенденту розраховується підсумкова когнітивна оцінка контролю знань, у кількості балів, яку розраховуємо за формулою (3):

$$Y = \sum_{k=1}^{n} Y_k , \qquad (2.3)$$

де Y_k – бал за відповідь на n-ну кількість питань.

За шкалою оцінювання підсумкових знань претендента виставляються такі оцінки:

- якщо $\frac{9}{10}n \leq Y \leq n$, претендент отримує оцінку «відмінно»;

- якщо $\frac{3}{4}n \leq Y < \frac{9}{10}n$, претендент отримує оцінку «добре»;

- якщо $\frac{3}{5}n \leq Y < \frac{3}{4}n$, претендент отримує оцінку «задовільно»;

- якщо $Y < \frac{3}{5}n$, за допомогою модуля адаптивного прийняття рішень проводиться аналіз тем адаптивної перевірки фахових знань, які претендент знає погано, і після їх додаткового опрацювання необхідне можливе повторне складання тесту.

Підсумковий контроль компетенцій претендентів у команди розробників комп'ютерних програмних систем відображає адаптивний варіант питань попередніх тестів. Провівши опрацювання отриманих даних адаптивної перевірки фахових знань претендентів у команди розробників програмних систем на базі автоматизованого програмного комплексу з тестування, програмний модуль виставляє підсумкову оцінку претенденту, яка фіксується як остаточна [41, 136, 247]. У випадку апеляції отриманої оцінки претендентом він має право на складання класичного тестування.

Розроблена модель адаптивного навчання й тестування передбачає виконання головної вимоги до контролюючої системи перевірки фахових знань, а саме гнучкість програми контролю та когнітивне охоплення предметної області системою питань.

Висновки до другого розділу

Наведені результати дослідження, які вивчали методи групової динаміки, їх вплив та проблеми реалізації, стратегії пом'якшення наслідків у галузі інженерії програмних систем з використанням системних емпіричних підходів. Командні керівники та менеджери можуть вибрати оптимальні методи при формуванні групової динаміки з врахуванням виявлених залежностей між методами і їх впливом. Також вони можуть стежити за результатами і визначити найкращі методи для команд певного типу, розміру і складу. У результаті емпіричних досліджень визначено найбільш важливі методи групової динаміки команди та наслідки, які пов'язані з використанням цих методів.

Відповідно до дослідження, що будь-яка група, а особливо розробників програмних систем, під час свого існування неминуче зіштовхується з груповими ефектами у різних формах розкриття. Перші групи повністю розкривають групові ефекти, другі просто оминають неприємні наслідки, а треті не в змозі пройти тернистий шлях формування та правильного управління – просто розпадаються. Отримані результати дослідження можуть бути корисними для керівників та

членів команд розробників програмних систем, що працюють над реальними проєктами.

Ґрунтуючись на системному аналізу у науковому досліджені в області формування команд з використанням визначених особистісних психотипів членів розробників програмних систем для проведення експерименту із доцільності застосування даних методів в галузі інформаційних технологій.

Проведений системний аналіз показав, що використання Agile software development технології під час розробки програмних систем є ефективнішим, аніж розробка без використання Agile. Окрім цього, використання методології групової динаміки Agile забезпечує вищий моральний дух команди, що призводить до підвищення продуктивності праці та дисципліни персоналу у команді розробників програмних систем.

Також у розділі проведено логіко-функціональний аналіз адаптивних моделей тестування знань претендентів у команди розробників програмних систем. Адаптивна система тестування частково дають змогу оцінити фахові знання отримані претендентами протягом підготовки до тестових випробувань завдяки реферативним та уточнюючим питань, що відображається у сумарній кількості отриманих ними балів.

Розроблено математичні методи планування й опрацювання результатів адаптивного тестування набутих компетенцій, із можливістю статистичної обробки даних. До переваг такої адаптивної інформаційної-аналітичної системи належить: можливість редагування банку тестів; зменшення навантаження на спеціалістів із набору персоналу завдяки передачі частини функцій адаптивній підсистемі контролю знань, а саме: швидке отримання результатів випробування і звільнення відповідних спеціалістів від трудомісткої роботи з опрацювання результатів тестування; об’єктивність в оцінці; конфіденційність при анонімному тестуванні; тестування на комп’ютері більш цікаве порівняно з традиційними формами опитування, що створює позитивну мотивацію в претендентів у групи розробників програмних систем. Структура інтелектуальної адаптивної підсистеми контролю знань є універсальною і не залежить від її наповнення.

Запропонована адаптивна система навчання й тестування може використовуватися для підвищення якості навчання на основі компетентного підходу. Застосування репозиторію тестових ресурсів дають змогу накопичувати тестові завдання та результати тестування для

необхідних компетенцій і на їх основі будувати індивідуальні тести, а також проводити їхній системний аналіз тестових питань. Застосування цієї системи сприяє адаптивному формуванню послідовності подання тестового матеріалу відповідно до поточних потреб компетенцій розробників програмних систем. Підсистему можна застосовувати без суттєвих змін у різних адаптивних інформаційних системах для аналізу компетентності претендентів у команди розробників програмних систем.

РОЗДІЛ 3

МАТЕМАТИЧНІ МОДЕЛІ ГРУПОВОЇ ДИНАМІКИ ФОРМУВАННЯ ТА ФУНКЦІОНУВАННЯ КОМАНД РОЗРОБНИКІВ ПРОГРАМНИХ СИСТЕМ

3.1. Теоретичні моделі формування та розвитку команд розробників програмних систем

Дамо визначення поняття «організація» – це організаційна система як об'єднання людей, спільно реалізують деяку програму або мету і діють на основі певних процедур та правил [2, 116-118]. Ці процедури і правила називаються механізмом функціонування фірм розробників програмних систем. Відзначимо, що саме наявність процедур і правил, що регламентують спільну діяльність членів команд в фірмі, є визначальною властивістю, яка відрізняє фірму від групи, команди та колективу (рис. 3.1).

Рисунок 3.1 – Моделі організаційної структури формування команд розробників програмних систем

Наведемо низку поширених визначень та термінів:

команда – це колектив, який здатний досягати мети автономно та узгоджено за мінімального управлінського втручання;

група – це сукупність людей, які об’єднані спільними інтересами, професією, діяльністю тощо;

колектив – це об’єднання людей, які згуртовані навколо спільних завдань;

організаційна структура (організація) – це об’єднання людей, які спільно реалізують деяку програмну або ціль та діють на основі визначених процедур та правил.

Формування та функціонування членів команд розробників програмних систем. На перший погляд потрібно розділяти два тимчасових етапи існування команди – формування і функціонування. Формування команди може бути водночас поділене на формування складу команди та її адаптацію, після чого можливий вже етап «стаціонарного» функціонування професійного складу команд розробників (рис. 3.2).

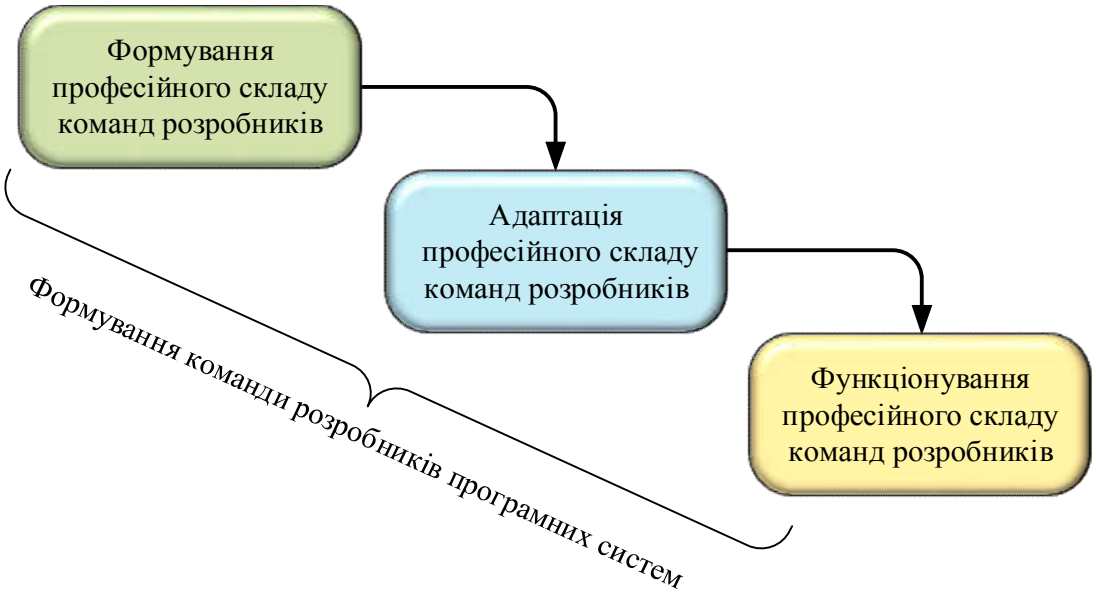


Рисунок 3.2 – Формування команд розробників програмних систем

Залежно від математичного апарату моделювання можна виділити кілька напрямів досліджень (рис. 3.3):

– завдання з призначення у команду розробників, що використовують переважно апарат оптимізації для виконання задач формування складу команд та розподілу ролей і обсягів робіт;

– теоретико-ігрові моделі, що застосовують апарат теорії ігор для опису і дослідження процесів формування та функціонування команд;

– рефлексивні моделі, що застосовують апарат теорії рефлексивних ігор для опису взаємодії членів команди, уявлення яких один про одного не збігаються;

– експериментальні дослідження команд розробників програмних систем, що охоплюють імітаційні експерименти та ділові ігри.

Саме цьому порівняно молодому напряму і присвячене це детальне дослідження, причому розглядаючи нові (і розвиток відомих) моделей значний акцент робиться на взаємній інформованості членів команд розробників програмних систем із використанням хмарних технологій.

Рисунок 3.3 – Класифікація моделей команд розробників програмних систем

Загальна модель організаційної системи формування команди розробників програмних систем охоплює:

– безліч членів команд (склад системи);

– технологічні, інформаційні, матеріальні тощо, а також зв'язки між ними у межах функціонування структури системи;

– безлічі допустимих дій членів команд системи, що відображають наявні фізичні, технологічні, нормативні та інші обмеження на стани дій, у яких вони можуть перебувати (вибирати самостійно) члени системи;

– цільові функції (що описують інтереси і переваги учасників), прагнення до максимізації яких відображає раціональність поведінки членів системи;

– інформованість та порядок функціонування.

Діяльність команди розробників програмних систем на основі хмарних технологій суттєво регламентується установленими нормами. У командах, на відміну від організацій, відсутня формальна ієрархія. Розглянемо основні характеристики команди розробників програмних систем, які відрізняють її від групи, колективу або організації (табл. 3.1):

– спільна мета;

– спільна праця над проєктом;

– спільне розуміння та сприйняття інтересів;

– автономність діяльності з розробки програмної системи;

– колективна та взаємна відповідальність за результати спільної діяльності;

– спеціалізація та взаємозамінність ролей (а також оптимальний розподіл функцій і обсягів та синергетичність у взаємодії членів команди);

– рівновагу команді розробників програмних систем (визначеність взаємних очікувань її членів).

Оскільки команди розробників програмних систем за своїм визначенням автономні, то, на відміну від теорії управління організаційними системами, при розгляді моделей команд керівний орган – центр (principal) окремо не виділяється. Що, однак, не заважає інтерпретувати цільову функцію або ефективність команди загалом як цільову функцію деякого центру, вирішального виконання завдань управління командою її формування та функціонування [6, 7, 9]. Умовно можна вважати, що безліч допустимих дій відображають «хто що може», цільові функції – «хто чого хоче», інформованість – «хто що знає». За аналогією з цими визначеннями можна виділити такі компоненти практично будь-якої моделі команд розробників програмних систем:

1. Склад команди (безліч членів, що входять у команду). Для того щоб описати команду, потрібно щонайменше задати її склад. У більшості моделей склад команди розробників програмних систем вважається фіксованим, хоча існують інші дослідження «команд» із динамічно змінним складом.

2. Стани членів команд (разом із функціями, які вони виконують (ролі) й обсяг завдань) мають безліч допустимих станів. Іноді в описі моделі є формули, що відображають взаємозв'язок між станами членів команд та закони мінливості станів у часі.

3. Залежно від того, чи вибирають члени команд свої стани самостійно або вважається, що вони визначаються ззовні (в наслідок розв'язання оптимізаційних завдань або встановлені певним керівним органом), виділяють відповідно моделі, що враховують активність членів команд, та моделі з пасивними членами команд. Активність поведінки членів команд розробників програмних систем зазвичай описується у межах теоретико-ігрових моделей.

Таблиця 3.1 – Математичні моделі, характеристики команд розробників програмних систем

Модель	Характеристика моделі						
	Спільна мета	Спільна діяльність	Спільність інтересів	Автономність дія-льності	Колективна та інди-відуальна відпові-дальність	Спеціалізація та взаємозамінність	Рівновага команди
Розподіл обсягу робіт	+	.				.	
Розподіл функцій	.	+				+	
Формування команди	+	.	.			.	
Синергетичний ефект	+	.	.			+	.
Модель Маршака-Раднера	+	+	+			.	
Стимулювання у командах	+	+	.	+	+	.	
Інституційне управління	.	+	.	.	+		.
Репутація	.	.	.	+	.	.	+
Експериментальні дослідження	.	+	.				.
Рефлексивні моделі однорідних та неоднорідних команд	.	+	.	+	.		+
Автономне прийняття рішень	.	.	.	+	.	+	
Розподіл витрат	.	.		.	.	+	+
Адаптація та самоорганізація у командах	.	.	.	+	.		+
Ієрархія у прийнятті рішень	.	.		.	.	+	

4. Результат діяльності команди розробників програмних систем, який залежить від станів членів команд (їхніх індивідуальних дій).

5. Цільові функції членів команд розробників програмних систем можуть залежати від їхніх індивідуальних дій (станів) або результату спільної діяльності. Причому цільові функції різних агентів можуть як збігатися (тоді є одна цільова функція, яка відображає єдину ефективність команди), так і відрізнятися.

6. Інформованість членів команд розробників програмних систем (інформація, якою вони мають володіють при значних зовнішніх і внутрішніх впливах) може бути як однакова, так і різна. Крім того, вона може бути тривіальною (коли є загальне знання – факт, за якого всім відомо, що всім це відомо або нетривіальною (у такому випадку необхідно враховувати ефекти рефлексії – уявлення членів команд про уявленнях один про одного).

Зв'язок між моделями команд з огляду на властивості їхніх членів команд встановлюється в табл. 3.2.

Таблиця 3.2 – Математичні моделі з огляду на характеристики команд розробників програмних систем

Модель	**Властивості членів команд розробників програмних систем**					
	Різноманіття завдань	**Активність членів**	**Різноманіття інтересів**	**Ступінь інформованості членів**	**Нетривіальна інформованість членів**	**Наявність динамічних змін**
Розподіл обсягу завдань						
Функціональний розподіл	+					
Формування команди	+	+	+			
Синергетичний ефект		+	+			
Модель Маршака-Раднера		+		+		
Стимулювання у командах розробників		+	+			
Інституційне управління командами		+	+	□		
Репутація команди розробників		+	+	□		
Експериментальне дослідження	+	+	+			
Однорідна команда розробників		+	+	□	+	□
Неоднорідна команда розробників	+	□	+	□	+	□
Автономне прийняття рішень		+	+	+	□	

Розподіл витрат	□	+	+			
Адаптація у командах	□	+		+	□	+
Навчання у командах	□	□				+

3.2. Моделі розподілу функціональних обов’язків у межах команд розробників програмних систем

У дослідженні розглядається термін «завдання про призначення», який є умовним і охоплює широкий клас оптимізаційних задач, що охоплюють завдання формування професійного складу команд, завдання розподілу функціональних обов’язків (ролей) у неоднорідних командах і завдання розподілу обсягу виконання завдань у межах команди [19].

Перераховані три типи завдань взаємопов'язані і вирішуються «циклічно». Отже, для того, щоб сформувати склад команди розробників програмних систем, потрібно знати, які функції буде виконувати той чи інший член, залучений до команди; а для оптимального розподілу функцій потрібно знати, який обсяг завдань доцільно виконати цьому члену в межах тієї чи іншої функціональності (рис. 3.4) [29, 30]. Тому розглянемо послідовно завдання розподілу обсягу робіт, завдання розподілу функцій і формування складу команди.

Рисунок 3.4 – Життєвий цикл взаємозв'язку завдань формування складу команд, розподілу функцій і розподілу обсягів між розробниками програмних систем

3.2.1. Рефлексивна модель формування однорідної команди розробників програмних систем

Проаналізуємо рефлексивну модель формування команди розробників програмної системи на основі хмарних технологій. Для цього

розглянемо множину $N = \{1, 2, \ldots, n\}$ членів команди. Стратегією вибору i-члена є вибір дій $y_i \geq 0$, що вимагає від нього затрат Кобба-Дугласа $c_i(y_i, r_i) = r_i \varphi(y_i / r_i)$ де $r_i > 0$ – тип члена, який визначає ефективність його діяльності; $\varphi(\cdot)$ – монотонно випукла функція [28, 32]. Якщо припустити, що метою спільної роботи членів команд розробників програмних систем є сумарна реалізація заданої поставленої задачі з мінімальними сумарними витратами на розробку.

$$\sum_{i \in N} c_i(y_i, r_i) \to \min_{(y_1, y_2, \ldots, y_n)} . \quad (3.1)$$

$$\sum_{i \in N} y_i = R . \quad (3.2)$$

У межах своїх уявлень кожен член команди розробників програмних систем може передбачити, які дії виберуть інші члени команди, якими будуть індивідуальні та сумарні витрати. Якщо вибір дій проводиться багаторазово, та спостерігається вибір дій деяким членом команди, який є відмінний від його уявлень, то він змушений буде коригувати свої уявлення, а при черговому своєму виборі використовувати «новий» набір уявлень [36, 40, 44]. Розглянемо два випадки структур інформованості (уявлення r_y та r_{yk}) і п'ять варіантів суб'єктивних історій гри, де будемо вважати, що суб'єктивні історії та структури інформованості всіх членів однакові, в іншому випадку число можливих варіантів вибору різко зростає, що водночас спричиняє появу десять моделей (табл. 3.3).

Таблиця 3.3 – Модель формування однорідної команди розробників програмних систем

Суб'єктивна історія моделі	Структура інформативності	
	$\{r_{ij}\}$	$\{r_{ijk}\}$
y_i	Модель 1	Модель 6
c_{-i}	Модель 2	Модель 7
c	Модель 3	Модель 8
$(y_{-i}; c_i)$	Модель 4	Модель 9
$(y_{-i}; c)$	Модель 5	Модель 10

Припустимо, що *i*-й член команди розробників програмних систем має структуру інформованості $\{r_{ij}\}$, спостерігаючи за діями інших членів. У такому випадку інформаційна рівновага буде мати таку математичну функцію:

$$y_i^*(\{r_{ij}\}) = \frac{r_{ij}}{\sum_{j \in N} r_{ij}}, \quad i \in N \quad . \tag{3.3}$$

Раціоналізоване уявлення набудуть такого математичного представлення:

$$\Omega_i^1 = \{r_{ij}\} > 0, j \in N \backslash \{i\} | r_{ij} / \sum_{j \in N} r_{ij} = x_j, \quad i \in N \backslash \{i\}\} \quad . \tag{3.4}$$

Динаміка уявлень *i*-го члена команди розробників програмних систем набуде такого вигляду:

$$y_{ij}^{t+1} = r_{ij}^t + \gamma_{ij}^t \left(w_{ij}^t(x_{-i}^t) - r_{ij}^t\right), \frac{r_{ij}}{\sum_{j \in N} r_{ij}}, \quad i \in N \backslash \{i\}, t = 1,2,\dots, i \in N, \tag{3.5}$$

де $w_{ij}^t(x_{-i}^t) - j$ - проекція найближчої k (r_{ij}^t), $i \in N \backslash \{i\}$ – точки множини Ω_i^1.

Множину стабільних інформаційних рівноваг наведено на графіку, в якому точкою позначено початковий стан, ромбиком – істинні значення психотипів, а стрілкою вказано зміну уявлень членів команд розробників програмних систем (рис. 3.5).

Рисунок 3.5 – Графік розподілу множини інформаційних рівноваг

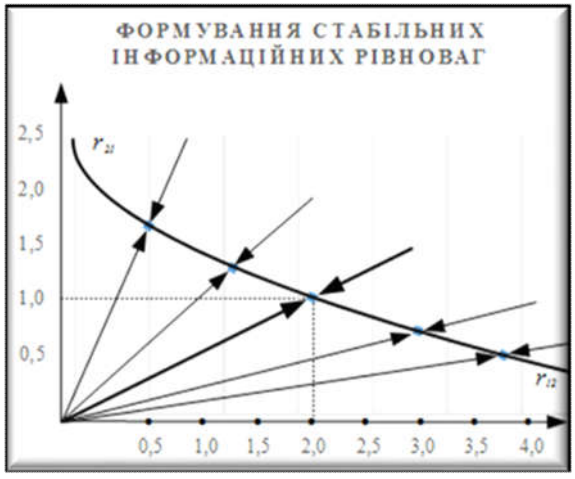


Рисунок 3.6 – Графік вузлів узгодженості стабільних інформаційних рівноваг

Рівняння стабільності при (n = 2)

$$r_{12}r_{21} = r_1 r_2, \quad (3.6)$$

За змістом математична умова (3.6) означає, що наскільки перший член команди розробників програмних систем переоцінює (недооцінює) іншого члена команди, на противагу у стільки разів другий недооцінює (переоцінює) першого. Агрегованою характеристикою команди розробників програмних систем загалом у цілому випадку можна умовно вважати добутком усіх членів команди [49, 74, 77].

Множина взаємних уявлень $(r_{12}; r_{21})$, що задовольняє математичну її умову (3.6), – це вид гіперболи на відповідній площині. Приклад випадку $(r_1 = 2; r_2 = 1)$ наведено на рис. 3.5.

Проведений математичний аналіз дає змогу не тільки визначити безліч рівноваг згідно з умовою (3.6), а й досліджувати області їхнього тяжіння: з умови (3.5), отже, випливає, що динаміка взаємних уявлень відповідає такому рівнянню:

$$\frac{\Delta r_{12}^t}{\Delta r_{21}^t} = \frac{\gamma_{12}^t}{\gamma_{21}^t}\,\frac{\gamma_{12}^{t-1}}{\gamma_{21}^{t-1}}, \mathrm{e} = 1,2,\dots,. \quad (3.7)$$

Відповідно до математичної умови рівняння (3.7), за постійних і однакових «кроків» γ траєкторіями зміни взаємних уявлень членів команди розробників програмних систем будуть прямі, що проходять через нуль. Кути нахилу цих прямих представлені на рис 3.6., де області тяжіння точок їх перетину з гіперболою умова (3.6), які визначаються початковою точкою (наприклад, будь-яка початкова точка лежить на виділеній на рис. 3.6 потовщеною прямою лінією $r_{12} = r_{21}/2$, тому виконується істинна рівновага).

Науковий погляд на це математичне представлення проявляє цікавий інтерес з погляду інформаційного управління, відповідно до того, що ми знаємо його кінцеву точку, центр легко може обчислити безліч початкових точок, почавши рух із якої члени команд розробників програмних систем самі знайдуть необхідну для центру рівновагу. Завершивши розгляд прикладу, можна зробити висновок, що стабільність команди розробників програмних систем та їхня злагодженість у виконанні поставлених завдань може досягатися, зокрема, за помилкових уявлень членів команди один про одного. Вихід із помилкової рівноваги вимагає отримання членами команд розробників програмних систем додаткового кванта інформації один про одного [45, 84, 86].

Отже, моделі формування та діяльності однорідних команд розробників програмних систем, що описуються термінами рефлексивних ігор, дають змогу ставити і вирішувати завдання управління процесом формування висококваліфікованих команд.

Дійсно, з розгляду умов моделей (3.1 – 3.7) випливає, що істотною є та інформація, яку мають члени команд розробників програмних систем про історію гри. Тому одна з управлінських можливостей полягає у створенні, по-перше, різноманітних ситуацій виробничої діяльності (що забезпечують виявлення істотних характеристик членів команд) і, по-друге, забезпечення максимальних комунікацій та доступу до всієї істотної інформації [88].

Крім того, проведений математичний аналіз свідчить, що на швидкість формування професійно збалансованої команди (швидкість збіжності до рівноваги) істотно впливають параметри γ та розміри ітерацій, які фігурують у процедурах динаміки колективної поведінки членів команд розробників програмних систем. Вплив на ці параметри також може розглядатися як управління з боку центральних органів на процес розробки програмної системи [85, 89, 90].

Отже, розглянуті «рефлексивні» моделі формування та функціонування професійних команд розробників програмних систем достатньо відображають такі властивості , як автономність, узгодженість і стійкість взаємодії різних членів команди.

3.2.2. Модель розподілу витрат на розв'язання завдання з розроблення програмних систем

Модель однорідної команди розробників програмної системи, що використовує єдиний програмно-технічний ресурс, сумарні витрати на придбання якого залежать від обсягу дій, які обирають члени команди. Умовою стійкого функціонування команди вважається існування такої процедури розподілу ресурсу, за якої можливий вибір членами такого вектора ненульових дій, який був би одночасно стійкий за Нешем (стійкий щодо індивідуальних відхилень його членів) і ефективний за Парето (вигідний для команди загалом) [92, 95, 100].

Отже, основні результати розподілу витрат на ресурси полягають у тому, що:

– відомо, що за плоских процедур розподілу витрат стійке функціонування команди неможливе;

– доведено, що якщо члени однорідної команди такі, що їх можна впорядкувати за ефективністю діяльності і воно не залежить від «обсягів виробництва», то стійке функціонування команди також неможливе;

– обґрунтовано, що умовою сталого функціонування команди є наявність синергетичної взаємодії її членів.

Модель:

$$f_i(x, r_i) = h_i(x_i, r_i) - \lambda_i(x), \qquad i \in N\,. \tag{3.8}$$

$$X = \sum_{i \in N} x_i\,. \tag{3.9}$$

де $C(X)$– витрати,

Припущення:

$\forall\, i \in N,\ \forall\, x \in \mathcal{R}_+^n\ \ \lambda_i(x) \geq 0.$

$\forall\, i \in N,\ \forall\, x \in \mathcal{R}_+^n\ \ \lambda_i(x)$ не зменшується за x_i.

$\forall\, x \in \mathcal{R}_+^n\ \ \sum_{i \in N} \lambda_i(x) = C(X).$

$\forall\, i \in N,\ \forall\, r_i \in \Omega_i\ \ h_i(0, r_i) = 0.$

$\forall\, i \in N,\ \forall\, x_{-i} \in \mathcal{R}_+^{n-1}\ \ \lambda_i(x_{-i}) = 0.$

$C(\cdot)$ незбіжна функція $C(0) = 0$.

Функція доходу та функція витрат – лінійні

Задача: $\exists\, \lambda(\cdot) - ?:\ \forall\, r \in \Omega\ \ P(r) \subseteq E_n(\lambda(\cdot), r).$

Умова наявності синергетичного ефекту:

$$\max_{x \in \mathcal{R}_+^n} \left[\sum_{i \in N} h_i(x_i, r_i) - C(X)\right] \geq \sum_{i \in N} \max_{y_i \geq 0} [h_i(y_i, r_i) - C(y_i)]. \tag{3.10}$$

Отже, на базі вище наведених математичних моделей можемо відзначити, що при використанні процедури розподілу витрат взаємна інформованість членів команд розробників програмних систем неважлива, а вектор типів членів команд немає бути всім відомим, оскільки «умова стійкості» для кожного члена команди містить тільки його психотип. Умова наявності синергетичного ефекту (3.10) охоплює типи усіх інших членів команд, проте вона повинна піддаватися перевірці, раніше, на етапі синтезу механізму розподілу витрат (створення умов діяльності команди розробників програмних систем) і вимагає лише потрібне знання істинного вектора типів членів команд

розробників програмних систем, не опираючись на будь яку рефлексію [64, 71, 101, 102].

Отже, основні результати дослідження процедур розподілу витрат полягають у тому, що:

– по-перше, показано, що за лінійних процедур розподілу витрат стійке функціонування команди неможливе;

– по-друге, доведено, що, якщо члени однорідної команди такі, що їх можна впорядкувати за паливною ефективністю діяльності, воно не залежить від «обсягів завдань», то стале функціонування команди також неможливе (наявність абсолютних лідерів руйнує «однорідну» команду);

– по-третє, обґрунтовано, що умовою сталого функціонування команди є наявність синергетичної взаємодії її членів.

3.2.3. Модель ефективної адаптації команд розробників програмних систем

Як зазначалося вище, одна з ключових відмінностей команд від організацій полягає в тому, що в перших, незважаючи на присутність лідера (як правило, неформального), відсутня формальна ієрархія. Розглянемо моделі самостійної адаптації команд до динамічно змінних умов. Адаптація тісно пов'язана з саморозвитком і самоорганізацією. Під саморозвитком розуміється напрям самовдосконалення фахових компетенцій, пов'язаних із переходом на вищий рівень функціонування фірми [106, 217, 218, 223]. Самовдосконалення – це розуміється зміна професійного стану члена команди розробників програмних систем під впливом внутрішньо властивих йому протиріч, факторів та умов. Водночас зовнішні впливи відіграють модифікаційну та опосередковану роль.

Більш загальним поняттям самовдосконалення є самоорганізація процесу розробки програмних систем, під час якого створюється, відтворюється або вдосконалюється структура складної інформаційної системи [108]. Відзначимо, що явище самостійного вибору членами команд та виконаних ними функцій, об’ємів завдань описаних вище. Моделі можуть інтерпретуватися як самоорганізація команди розробників програмних систем на відміну від централізованої організації діяльності, що здійснюється в ієрархічних організаційних системах керівним органом.

Адаптація (від лат. Adaptatio – «пристосування») – пристосування до умов існування та адаптування до них; у соціальних системах вид взаємодії із середовищем, під час якого узгоджуються вимоги й очікування всіх членів команд. У межах моделей формування команд розробників програмних систем із використанням хмарних технологій під адаптацією будемо розуміти процес зміни дій (включаючи у загальному випадку функції і обсяги виконаних завдань), які обирають члени команди на основі актуальної інформації в умовах, що динамічно змінюються (рис. 3.7) [109].

На практиці можна виділити декілька вкладених рівнів адаптації членів команд будь-якої програмної системи:

– зміна інформованості;
– зміна поведінки як дій, які обираються на основі наявної інформації;
– зміна параметрів системи, що дає змогу реалізовувати більш ефективну поведінку за зміни умов;

цілеспрямована зміна зовнішнього оточення (активна адаптація).

Рисунок 3.7 – Рівні адаптації команд розробників програмних систем

Члени команди розробників програмних систем раціональні, їхні інтереси описуються цільовими функціями, а раціональність поведінки кожного члена команди полягає у прагненні максимізувати свою цільову функцію, проте на кожній ітерації часу приймаються свої відпрацьовані рішення в умовах неповної інформованості [111]. З часом вони накопичують інформацію про нечітко невизначені парамет-

ри, а також можуть бути різні «стратегії» поведінки членів команд розробників з погляду тих цілей, які вони мають.

3.2.4. Формування структурної моделі для адаптації команд розробників

Специфіка команд розробників програмних систем полягає, зокрема, і в тому, що кожен член команди як інформацію для корегування своїх уявлень про невизначені параметри може використовувати не тільки результати спостереження за зовнішнім середовищем, а й висновки щодо дій та інших членів команд, намагаючись «пояснити», чому вони вибрали саме цю стратегію дій (рис. 3.8).

Рисунок 3.8 – Структура моделі адаптаційного періоду команд розробників програмних систем

Визначимо інтервал часу, упродовж якого відбувається загальна адаптація членів команди до позитивних стосунків між нами. $\pi(x) = \{\theta \in \Omega | \exists\, \Omega_0 : \theta \in \Omega_0, x \in E(\Omega_0)\}$ – множина станів природи, за яких досліджуваний вектор члена команди розробників програмних систем є зрівноваженим. Отже, $g = (g_1, g_2, \dots, g_n) \in \mathcal{R}^n$ – є вектор, який досліджується як властивість члена команди розробників програмних систем, та значення його цільової функцій. Тому математичною умовою є $\delta(g, z) = \{\theta \in \Omega | f_j(\theta, z) = g_j, j \in N\}$ – множина тих значень станів навколишнього світу, за яких із результатом z можуть бу-

ти позитивно реалізовані виграшні стратегії члена команди, який досліджується.

Отже, $p(x,z) = \{\theta \in \Omega \,|G(\theta,x) = z\}-$ де $p \subseteq \Omega-$ множина значень станів природи, за яких спостережуваний вектор дій члена команди розробників програмних систем приводить саме до цього значення результату z яке досліджується.

У i-го члена команди розробників програмних систем є, щонайменше чотири незалежних джерела інформації про стан природи:

апріорна особиста інформація $w_i(\theta) \subseteq \Omega$;

дії інших членів команд розробників програмних систем спостерігаючи за ними та передбачаючи, що вони будуть діяти раціонально. У такому випадку член команди може (вважаючи, що є загальне правильне знання на першому рівні структури інформованості) здійснювати рефлексію оцінюючи отримані відомості $p(x)$ про стан досліджуваної природи, на підставі якої і був раціональний вибір колегами саме цих дій;

успішне виконання тих чи інших завдань членів g команд розробників програмних систем, які діють на підставі цієї інформації можуть зробити адекватний висновок $\delta(g,z)$ про ті чи інші стани природи, за яких сумарний результат сприяє появі спостережуваних позитивних результатів виконання поставленого завдання;

результат спільної діяльності команди розробників програмних систем з використанням хмарних технологій, який саме сприяє появі бажаного результату $p(x,z) = \{\theta \epsilon \Omega \mid G(\theta,x) = z\}$.

Відзначимо, що в через введені припущення інформація пунктів з 2-4-го є загальним знанням серед членів команд, тобто з погляду один одного вони, спостерігаючи одні і ті ж самі інформаційні події, повинні однаково (і передбачувано для опонентів) змінювати свої уявлення про виробничі оточення. Тобто, загалом отриманими знаннями є інформація, наведена у виразі:

$$I(x,z,g) = \pi(x) \cap p(x,z) \cap \delta(g,z) \subseteq \Omega. \qquad (3.11)$$

Цим математичним припущенням, поряд з припущеннями про те, що кожен член команди вважає, що є загальне знання на першому рівні структури інформованості, вилучається з розгляду (але не з предметів подальших досліджень) рефлексія агентів щодо інформованості опонентів.

3.3. Динамічні моделі адаптації команд розробників програмних систем

Можливість «повторного» використання інформації, отриманої внаслідок спостереження за результатами діяльності інших членів команд розробників програмних систем, з'являється, якщо член команди повторює вибір своєї дії [116, 121]. Будемо вважати, що члени команд вибирають свої дії на кожному кроці одночасно, а інтервали часу є рівномірними.

Отже, $x_i^t \in X_i$ – виконання завдань *i*-го члена команди розробників програмних систем у певний момент часу *t*, $ar^{1,t}$ – синергетична дія об'єднань векторів усіх членів команди розробників програмних систем за фіксований час *t* періодів (квантів часу).

До кінця загального періоду *t* зваженим значенням інформації для членів команди розробників програмних систем є інформація:

$$I(x^t, z^t, g^t) = \pi(x^t) \cap p(x^t, z^t) \cap \delta(g^t, z^t) \subseteq \Omega. \tag{3.12}$$

На наступному етапі проведемо оцінку членом команди розробників програмних систем з використанням хмарних технологій загальних станів виконання завдань:

$$I_i^t(\omega_i, x^{1,t}, z^{1,t}, g^{1,t} = \omega_i \cap \bigcap_{\tau=1}^{t} I(x^\tau, z^\tau, g^\tau). \tag{3.13}$$

Приклад випадку моделі олігополії Курно:
нехай

$$n = 2,\ x_i \geq 0,\ i = 1, 2, \ldots, n,\ z = x_1 + x_2, \Omega = [1; 4]; \tag{3.14}$$

$$\omega_i = [2, 5];\ \theta_j = 3, f_j(\theta, z) = (\theta - \alpha z)z - x_i^2 r\ /2, \tag{3.15}$$

де $\alpha > 0, r > 0$ – відомі розмірні константи. Якщо припустити, що кожен член команди розробників програмних систем з використанням хмарних технологій спостерігає лише своїми діями та націлений тільки на позитивний результат від своєї діяльності, і якщо б значення об'єктивного стану для виграшу було відомо, то йому необхідно було б вибрати тому послідовність дій:

$$x_i^*(\theta) = \frac{\theta}{4\alpha + r}, \qquad i = 1, 2, \dots, n. \tag{3.16}$$

У такому випадку стани поведінки членів команд розробників систем набудуть такого вигляду:

$$\theta_i^t = \frac{\left(\theta_0 - \alpha(x_1^t, x_2^t)\right)(x_1^t, x_2^t)}{2x_i^t} + 2\alpha x_1^t, \qquad i = 1, 2, \dots, n, t = 1, 2, \dots, m. \tag{3.17}$$

Ґрунтуючись на вищенаведених теоретичних положеннях, члени команд розробників програмних систем з використанням хмарних технологій будуть вибирати такі шляхи прийняття відповідних виробничих рішень:

$$x_i^t\left(\theta_i^{t-1}\right) = \frac{\theta_i^{t-1}}{4\alpha + r}, \qquad i = 1, 2, \dots, n, t = 1, 2, \dots, m \tag{3.18}$$

Час адаптації команди розробників програмних систем з використанням хмарних технологій є саме той час, за який при незмінній структурі та особистому складі команди спостерігається позитивна інформація, за якою можна однозначно або із заданою апріорною точністю ідентифікувати внутрішній стан команди як позитивний [122 ,128, 129]. Сумарне значення часу адаптації членів команди насамперед залежить від того, за якими саме параметром необхідно вести спостереження та якої розмірності вектор, який описує консолідований внутрішній стан команди розробників, а також властивості точково-множинних відображень $\pi(*)$, $p(*)$, $\delta(*)$ (рис. 3.9, 3.10).

динаміка траєкторії адаптації першого члена команди

динаміка траєкторії адаптації другого члена команди

Рисунок 3.9 – Динаміка адаптації команд розробників програмних систем

динаміка траєкторії поведінки при збурені першого члена команди

динаміка траєкторії поведінки при збурені другого члена команди

Рисунок 3.10 – Динаміка траєкторії часової характеристики адаптації команди

Адаптація відповідає пристосуванню, звиканню до зовнішніх та внутрішніх умов праці. У дисертації розглянуто динамічні моделі адаптації команд, які дають змогу відображати ці ефекти [131, 133]. Наведемо приклад процесу динаміки адаптації команди до різкої зміни зовнішніх або внутрішніх умов. Припустимо, що в певний період часу (рис. 3.9) на 12-му кроці значення стану навколишнього середовища змінилося і стало одного члена команди не 4, та відповідно іншого також 4, причому початкові оцінки нового значення стану навколишнього оточення: у першого члена команди – 3,6, а у другого – 4,25. Динаміка траєкторії часової характеристики адаптації членів команд при збурені наведена на рис. 3.10, відповідно до значень станів навколишнього середовища.

У розглянутих динамічних моделях швидкість зміни станів навколишнього середовища щодо часу адаптації членів команд розробників програмних систем така, що команда «встигає» відстежувати зміни [140, 150]. Проте і можливі випадки – в умовах швидко плинної зміни станів навколишнього середовища, – коли команда розробників не зможе адаптуватися. Отже, підкреслимо припущення про те, що кожен член команди розробників програмних систем наділяє опонента інформаційними потоками, які він має.

Ґрунтуючись на наших дослідженнях та математичних моделях при розгляді більш складних структур інформованості членів команд, вважаємо, що вони будуть вибирати такі дії, які мають інформаційну рівновагу [153]. Отже, можливі ситуації зі складнішою структурою «спостережень» членів команд: одні можуть спостерігати за одними параметри, а інші члени команди – іншими параметрами.

Якщо динамічна адаптація розглядалася як пристосування до зміни значень станів навколишнього середовища, а саме існування та звикання до них, і фактично залежить від інформації про ці зміни. Звичайно, загалом випадку динамічна адаптація деякої програмної системи має на увазі не тільки зміну інформованості та поведінки, а й зміну параметрів самої системи. Крім того, можна розглядати й активну адаптацію, коли система цілеспрямовано впливає на зовнішнє середовище.

3.4. Модель набуття та вдосконалення фахових компетенцій у командах розробників

Інноваційні методи та засоби набуття і вдосконалення фахових компетенцій у командах розробників програмних систем є предметом досліджень у сфері інформаційних технологій, кібернетики та суміжних галузях упродовж тривалого часу. Крім того, команда розробників програмних систем є благодатний об'єкт для математичного моделювання на рівні аналогій. Для прикладу наведемо, комп'ютерну нейронну мережу і назвемо її «командою», членами якої є окремі нейрони [114]. Відомо безліч методів навчання нейронних мереж, і всіх їх можна інтерпретувати за допомогою термінів «навчання команд». Або є інший приклад: «командою» можна назвати GRID-систему, яка здійснює розподілені математичні обчислення. В іншому випадку нас будуть цікавити ефекти автономності активної поведінки членів команди розробників програмних систем, тому ми будемо акцентувати фахове навчання у процесі виконання завдань з оптимального погляду команди загалом та оптимальному розподілі обсягу робіт між її членами [152, 154].

Відповідно, виділяють результативні показники ітеративного навчання і характеристики адаптації, які належать зазвичай до фізіологічних компонентів діяльності членів команд. Ми розглядаємо, які саме результативні ітераційні методи навчання є найбільш ефективними [170]. Ітераційне навчання переважно характеризується такими сповільнено-асимптоматичними кривими навчання, які апроксимуються експонентними кривими:

$$r(t) = r^{\infty} + (r^{0} - r^{\infty})\, e^{-\gamma t}, t \geq 0 \qquad (3.19)$$

або дискретною послідовністю

$$r^{k} = r^{\infty} + (r^{0} - r^{\infty})\, e^{-\gamma t}, k = 1, 2, \ldots, n, \qquad (3.20)$$

де t – *час* навчання кожного члена команди розробників програмних систем на основі хмарних технологій; k – число ітерацій (спроб) з початку навчання; $r(t)$ та r^{k} – це тип члена команди (рівень професійних компетенцій, набутих кваліфікацій) у певний момент часу t (на k-ітерації); $r^{0} \geq 0$ – початкове значення, яке відповідає певному моменту початку навчання – першому інтервалу часу типу; r^{∞} – кінцеве

значення, де $r^{\infty} \geq r^{0}$, деяка невід'ємна константа, яка визначає швидкість зміни або корегування певного психотипу члена команди розробників програмних систем та називається швидкістю навчання (рис. 3.11).

Позначимо верхнім індексом номер періоду часу, а нижнім – номер члена команди розробників програмних систем. У разі, коли розглядається єдиний член команди, нижній індекс будемо опускати.

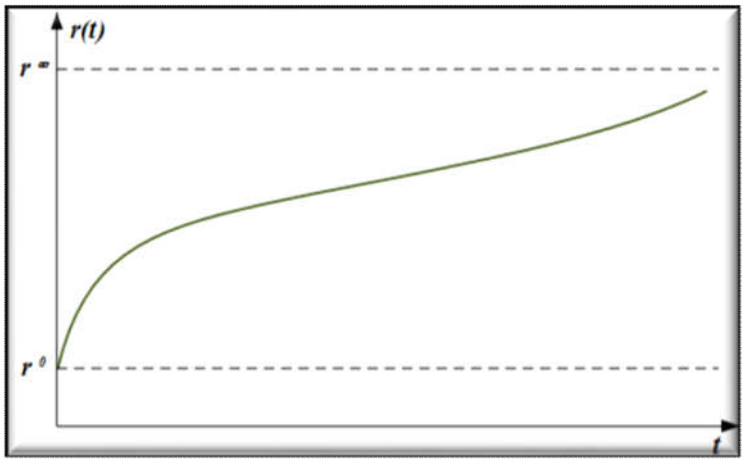


Рисунок 3.11 – Експоненціальна крива навчання члена команди розробників програмних систем

Позначимо $y^k \geq 0$ завдання, що виконує член команди розробників програмних систем у k – періоді часу виконання поставлених завдань. Якщо інтерпретувати психотипи членів, які виконують програмні завдання за рівнем професійних навиків, то $r^k \in [0,1]$ можна визначити як частку успішно виконаних дій членом команди розробників програмних систем, що виконують у певний період k обсяг завдань y^k, то пересічний член команди досягне певного результату $z^k = r^k y^k$. Тоді узагальнений результат кожного із членів команди розробників програмних систем з використанням хмарних технологій буде сумарним обсягом виконаних завдань, які він успішно виконав за k ітераційних періодів відрізків часу, так званих (тасків) буде дорівнювати:

$$Z^k = \sum_{i=1}^{k} r^i y^i \quad . \tag{3.21}$$

В іншому випадку кожен із членів команди розробників програмних систем з використанням хмарних технологій виконує велику кількість дрібних завдань, деякі із них мають успішне завершення, а інші – зовсім вдалі.

$$Y^k = \sum_{i=1}^{k} y^i. \tag{3.22}$$

Цей масив практичних завдань умовно можна зарахувати до набуття професійного досвіду та навиків, які опанував кожен із членів команди розробників програмних систем.

У такому випадку:

$$r^k = 1 - (1 - r^0)\, txp(-\gamma\, Y^{k-1}),\ \ k = 2, 3. \tag{3.23}$$

Визначимо динаміку обсягів успішно виконаних та вчасно завершених завдань і психотипів членів команд розробників програмних систем з використанням хмарних технологій згідно з таким математичним виразом:

$$Z^k = \sum_{i=1}^{k} y^i \left\{ 1 - (1 - r^o) \exp\left(-\gamma \sum_{m=1}^{i-1} y^m \right) \right\}, \tag{3.24}$$

$$r^k = 1 - (1 - r^0)\, txp\left(-\gamma \sum_{i=1}^{k-1} y^i \right). \tag{3.25}$$

$k = 2, 3, \ldots$.

Відзначимо, що за фіксованого сумарного обсягу робіт тип члена команди розробників програмних систем з використанням хмарних технологій визначається за допомогою виразу (3.23) однозначно і не залежить від того, як обсяг робіт розподілені за періодами часу. Тому вирішення завдання максимізації типу члена команди, а саме досягнення максимальної його кваліфікації за фіксованого сумарного обсягу робіт, Y^t у межах цієї моделі неможливе.

У моделі фігурує три «макропараметри»: сумарний обсяг робіт *Y*, число періодів *T* і результат *Z*. Досліджувана змінна відображає «траєкторію навчання» $y^{1,T}$.

Завдання оптимального навчання *i*-го члена команди розробників програмних систем полягає в екстремізації однієї із змінних за фіксо-

ваних інших змінних. Отже, ми визначаємо, що доцільно розглядати такі завдання:

Фіксуємо загальний обсяг завдань *Y,* який може виконувати пересічний член команди розробників програмних систем із результатом *Z,* який необхідно досягнути. Це завдання вимагає знаходження необхідної траєкторії, з мінімальними витратами по часу для досягнення поставленого результату:

$$\begin{cases} T \to min \\ Y^T \leq Y \\ Z^T \geq Z \end{cases} \tag{3.26}$$

Задачу (3.26) можна умовно назвати задачею про визначення швидкодії.

Фіксуємо загальний обсяг завдань *Y,* який може виконувати пересічний член команди розробників програмних систем із часом навчання *T.* Необхідно найти найбільш оптимальну траєкторію, яка б давала змогу максимізувати отриманий результат *Z.*

$$\begin{cases} Z^\tau \to max \\ Y^\tau \leq Y \\ \tau \leq T \end{cases} \tag{3.27}$$

Задачу (5.27) можливо умовно назвати задачею про оптимальне навчання або отримання фахових компетенцій. Зазвичай саме ця задача є достатньо близькою до проблем педагогічної галузі, оскільки коли необхідно за фіксований час та за фіксованого обсягу навчального матеріалу потрібно зробити такий розподіл у часі, щоб забезпечити максимальний об’єм вивченого матеріалу, тим самим максимізувати якість навчального процесу, що еквівалентно:

$$\sum_{l=1}^{T} y^l \exp\left(-\gamma \sum_{m=1}^{l-1} y^m\right) \to \frac{min}{\{y^{1,T} | \sum_{\tau-1}^{T} y^\tau = Y\}}. \tag{3.28}$$

Графічне представлення навчання та набуття фахових компетенцій, а також розподілу оптимального навантаження на членів команд розробників програмних систем з використанням хмарних технологій (рис. 3.12 – 3.14).

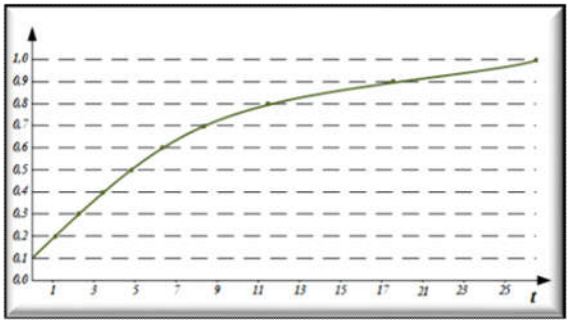


Рисунок 3.12 – Динаміка навчання *1*-го члена команди розробників програмних систем

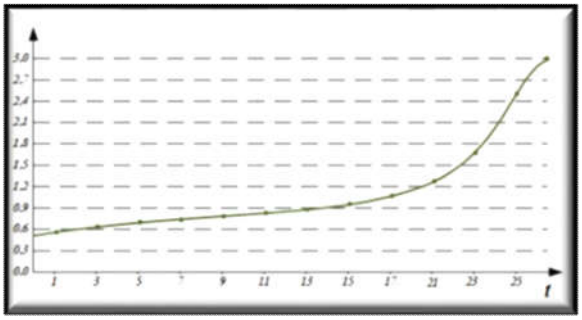


Рисунок 3.13 – Динаміка оптимального розподілу навантаження на *1*-го члена команди розробників програмних систем

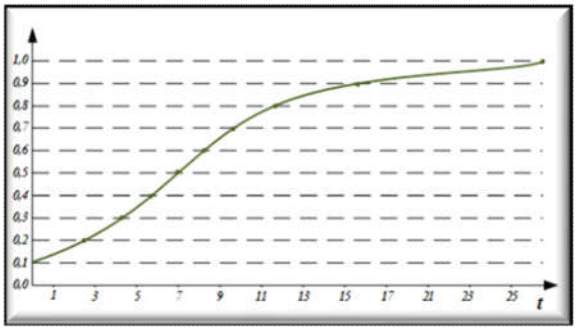


Рисунок 3.14 – Динаміка навчання іншого члена команди розробників програмних систем

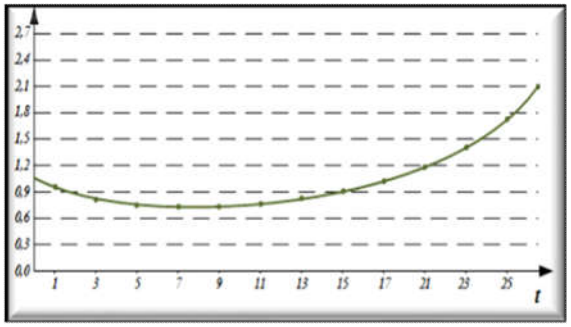


Рисунок 3.15 – Динаміка оптимального розподілу навантаження на іншого члена команди розробників програмних систем

Оптимальною стратегією навчання та набуття фахових компетенцій є збільшення обсягів виконання поставлених завдань з часом, причому чим вища швидкість навчання, тим більше опукліша є оптимальна траєкторія. Якщо крива навчання опукла, то член команди розробників програмних систем навчається дедалі ефективніше. Отже, оптимальна траєкторія навчання буде мати спадну траєкторію, тобто оптимальною стратегією навчання буде вже не збільшення, а зменшення обсягу завдань із згодом.

3.5. Модель сумісної взаємодії декількох членів команд розробників

Відповідно до теоретичних положень щодо одного члена команди розробників програмних систем з використанням хмарних технологій, розглянемо взаємодію декількох. Узагальнивши дані нашого

дослідження, розглянемо одночасно працюючих декількох членів команд розробників програмних систем, які працюють одночасно. Спочатку розглянемо ситуацію, коли члени команди повністю незалежні, тобто результати і психотип кожного із них не залежать від результатів та психотипів інших учасників команди. А на наступному етапі проаналізуємо виконання завдань про навчання та набуття фахових компетенцій взаємозалежних залежних членів команд розробників програмних систем [99, 175, 176, 214].

Опис моделі: $N = \{1,2,\dots,n\}$ – команда, що складається з n-членів.

За аналогією з виразами (3.24) і (3.25) отримаємо такі математичні вирази для відповідних обсягів успішно виконаних завдань членами команд:

$$Z_i^k = \sum_{l=1}^{k} y_i^l \left\{1 - (1 - r_i^0)\exp(-\gamma_i \sum_{m=1}^{l-1} y_i^m)\right\} \tag{3.29}$$

де Z_i^k – обсяги успішно виконаних робіт.

$$r_i^k = 1 - \left(1 - r_i^0\right)\exp\left(-\gamma_i \sum_{l=1}^{k-1} y_i^l\right), k = 2,3,\dots, i \in N. \tag{3.30}$$

де r_i^k – типи членів команд.

Отже, якщо результат команди є сумою результатів її членів, вираз набуде такого значення:

$$Z^k = \sum_{i=1}^{n} Z_i^k, k = 1,2,\dots, \tag{3.31}$$

тому поставлене завдання про оптимальне навчання та отримання фахових компетенцій декількох членів команд розробників програмних систем набуде такого вигляду:

$$Z^T \to \left\{y_i^{1,T} \Big| \sum_{\tau=1}^{T}\sum_{i=1}^{N} y_i^\tau = Y\right\}^{max}, \tag{3.32}$$

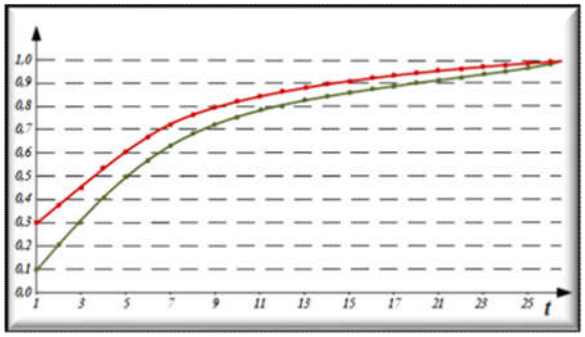
отже:

$$\sum_{i=1}^{n}\sum_{l=1}^{T} y_i^l \left\{1 - (1 - r_i^0)\exp(-\gamma_i \sum_{m=1}^{l-1} y_i^m)\right\} \to \left\{y_i^{1,T} \Big| \sum_{\tau=1}^{T}\sum_{i=1}^{N} y_i^\tau = Y\right\}^{max}. \tag{3.33}$$

Завдання (3.33) може бути виконане методом динамічного програмування. Зрозуміло, що оптимальне рішення задачі (3.33) загалом залежить і від індивідуальних швидкостей навчання та набуття фахових компетенцій членами команд $\{\gamma_i\}$, і від їхніх початкових кваліфікацій $\{r_i^0\}$.

Отже, відповідно до математичних моделей швидкості навчання та набуття фахових компетенцій членів команд розробників програмних систем з використанням хмарних технологій однакові, тому оптимальним розподілом виконання завдань є загальний обсяг робіт членів команд із максимальною початковою кваліфікацією. Якщо початкові кваліфікації членів команд приблизно однакові, то оптимальним розподілом завдань є виконання усього обсягу робіт членами команд із максимальною швидкістю навчання [177].

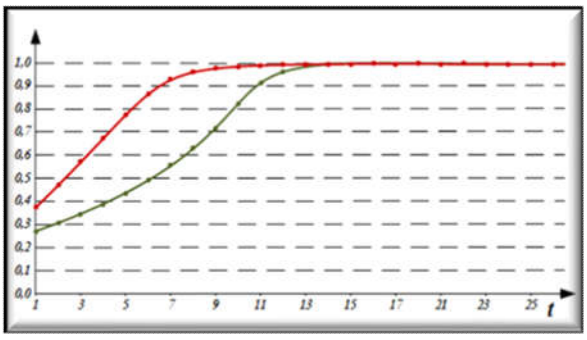
Якщо додати ще одного члена команди зі специфічним психотипом, то оптимальним вирішенням розподілу завдань буде сумарний обсяг робіт, які кожен член може виконати самостійно за фіксований інтервал часу, причому останнє обмеження є дійсне для усіх членів команд розробників програмних систем. Проте якщо у кожного члена команди розробників існують власні обмеження щодо обсягу виконаних завдань, який він може виконати за певний інтервал, то завантажені будуть уже усі члени команди, оскільки сумарне число обмежень буде більшим, ніж сама кількість членів команд (рис. 3.16, 3.17).



— *динаміка траєкторії психотипу першого члена команди*

— *динаміка траєкторії психотипу іншого члена команди*

Рисунок 3.16 – Динаміка психотипів членів команд



— *динаміка траєкторії оптимального навантаження першого члена команди*

— *динаміка траєкторії оптимального навантаження іншого члена команди*

Рисунок 3.17 – Динаміка оптимальних обсягів завдань членів команд

Відповідно до нашої гіпотези під час розгляду навчання членів команд у процесі виконання завдань ми вважали, що кожен член команди розробників програмних систем з використанням хмарних технологій вчиться тільки «на власному досвіді». Проте у командах має місце постійний обмін досвідом та фаховими навиками. Отже, члени команд мають змогу спостерігати за успіхами та труднощами у діяльності інших, а також у такий спосіб набувати відповідного фахового досвіду. Для того, щоб розглянути цей ефект, опишемо «досвід», накопичений *i*-м членом команди розробників програмних систем, не тільки як суму його власних дій, але й додаймо до цієї суми зважену суму дій інших членів команд [189, 208]. Відповідно до вище наведених положень, маємо такі математичні вирази для відповідних обсягів успішно виконаних завдань та психотипів членів команд розробників програмних систем:

$$Z_i^k = \sum_{l=1}^{k} y_i^l \left\{ 1 - (1 - r_i^0) \exp(-\gamma_i \sum_{j=1}^{n} \alpha_{ij} \sum_{m=1}^{l-1} y_j^m) \right\}, \tag{3.34}$$

$$r_i^k = 1 - \left(1 - r_i^0\right) \exp\left(-\gamma_i \sum_{j=1}^{n} \alpha_{ij} \sum_{l=1}^{k-1} y_j^l \right), k = 2,3,\dots, i \in N, \tag{3.35}$$

де константи $\{\alpha_{ij} \geq 0\}$ можуть інтерпретуватися як міра ефективності передачі досвіду від *j*-го члена команди розробників *i*-му, $i, j \in N$.

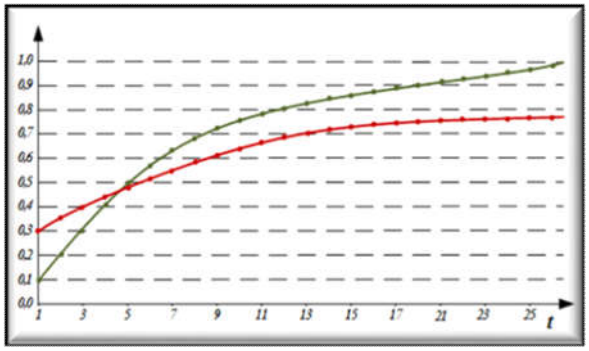
Тоді завдання про оптимальне навчання з огляду на обмін досвідом між членами команд розробників програмних систем набуде такого вигляду:

$$\sum_{i=1}^{n} \sum_{l=1}^{T} y_i^l \left\{ 1 - (1 - r_i^0) \exp(-\gamma_i \sum_{j=1}^{n} \alpha_{ij} \sum_{m=1}^{l-1} y_i^m) \right\} \rightarrow \max \left\{ y_i^{1,T} \mid \sum_{\tau=1}^{T} \sum_{i=1}^{N} y_i^\tau = Y \right\}. \tag{3.36}$$

Наприклад: нехай швидкості навчання двох членів команд однакові, проте другий член команди має кращі початкові компетенції та кваліфікацію при матриці $\|\alpha_{ij}\| = \begin{bmatrix} 1 & 2 \\ 0 & 1 \end{bmatrix}$. Отже, перший член команди навчається на своєму досвіді і на досвіді другого члена команди на-

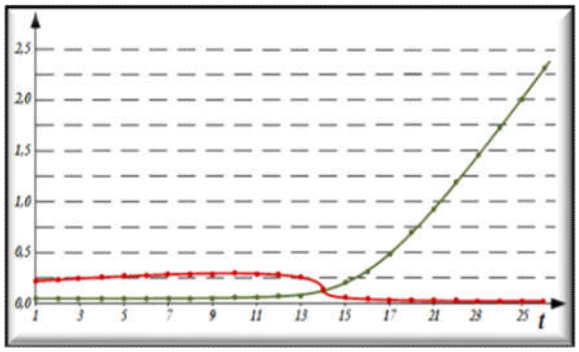
віть ефективніше, ніж на своєму. Другий же член команди розробників програмних систем навчається тільки на своєму досвіді.

Щоб оцінити навчання двох психотипів членів команд, розглянемо їхню динаміку за визначеного обсягу завдань (рис. 3.18, 3.19).



— *динаміка траєкторії психотипу першого члена команди*
— *динаміка траєкторії психотипу другого члена команди*

Рисунок 3.18 – Динаміка взаємодії психотипів членів команд



— *динаміка траєкторії оптимального об'єму навантаження першого члена команди*
— *динаміка траєкторії оптимального об'єму навантаження на другого члена команди*

Рисунок 3.19 – Динаміка розподілу оптимальних обсягів навантаження членів команд

Упродовж перших дванадцятьох періодів перший член команди розробників програмних систем практично не виконує завдання сам, а «спостерігає» за діями другого члена команди. Водночас фахова кваліфікація першого члена команди зростає набагато швидше, аніж другого члена команди.

Починаючи з тринадцятого періоду, оптимальним є виконання всього обсягу навантаження першим членом команди, а не другим членом команди. Отже, якщо ввести обмеження на обсяг виконаного навантаження, яке виконується членом команди за одиницю часу, то структура рішення, відповідно зміниться. Це графічне відображення у науковому досліджені наочно ілюструє, як недолік початкової кваліфікації може бути успішно компенсований ефективним навчанням на чужому досвіді членів команди розробників програмних систем [224]. Існує й інше інтерпретування цього випадку. Можна вважати другого члена команди учителем, тьютором або наставником, який, маючи вищу початкову фахову кваліфікацію, навчає першого члена команди. У певний момент перший член команди «обганяє» його і може працювати самостійно.

Відповідно до дослідження «рефлексивних моделей», можна зробити висновок, що найважливішою умовою стабільного й ефективного функціонування команди є наявність загальних фахових компетенцій та професійного досвіду. І саме тому на їхнє формування професійних знань зазвичай націлена більшість організаційних та інших зусиль у процесі формування та навчання команди [212, 220, 225, 233].

Аналізуючи результати досліджень математичних моделей взаємного навчання команд розробників програмних систем, можна провести певну градацію в отриманні фахових компетенцій при виконанні певного обсягу навантаження:

– за фіксованого сумарного обсягу робіт одного члена команди результативні характеристики навчання не залежать від того, як розподілені за часом;

– розв'язання задачі про оптимальне ітеративне навчання одного члена команди не залежить від його початкової кваліфікації;

– чим вища швидкість навчання члена команди, тим більший обсяг навантаження він повинен виконувати в останніх періодах (і, відповідно, тим менший обсяг навантаження необхідно виділяти на початкові періоди для підвищення його початкової кваліфікації);

– оптимальною стратегією ітеративного навчання є збільшення обсягу навантаження члена команди з часом, причому чим вища швидкість навчання, тим випуклішою є оптимальна траєкторія навчання. Якщо крива навчання випукла (член команди навчається все дедалі ефективніше), то оптимальна траєкторія навчання буде зменшуватися, тобто оптимальною стратегією навчання буде вже не збільшення, а зменшення обсягу навантаження члена команди з часом;

– якщо немає обмежень на індивідуальні обсяги навантаження, то в команді увесь обсяг виконує «кращий» (з погляду комбінації початкової кваліфікації і швидкості навчання) член команди;

– недостатня початкова кваліфікація члена команди може бути успішно компенсована ефективним навчанням як на його власному, так і на чужому досвіді;

– найважливіша умова стабільного й ефективного функціонування команди – наявність загального знання, на формування якого зазвичай націлена більшість організаційних і інших зусиль у процесі формування та навчання команди.

3.6. Модель професійного розвитку персоналу у команді розробників програмних систем

Ефективні інноваційні підходи до управління фірм вимагають кваліфікованих фахівців, одним із найважливіших критеріїв сучасних підходів до інноваційного розвитку фірм є перманентний розвиток її персоналу [240].

Якщо розглядати ефективне управління організаційною системою (ОС) у реаліях сьогоденних вимог, то в теорії управління організаційними системами можна виділити такі типи управління:

– управління складом членів команд розробників програмних систем з використанням хмарних технологій;

– управління структурою членів команди розробників;

– управління мотиваціями та вподобаннями членів команд розробників програмних систем;

– управління обмеженнями та нормами діяльності згідно з законодавчими документами;

– управління інформаційними потоками даних, які використовуються при прийнятті ключових рішень

Ґрунтуючись на такій класифікації управління інноваційним розвитком членів команд розробників програмних систем, можемо розглядати першочергово управління персональним складом організаційної системи [245, 249, 258]. Однак, при здійсненні такого інноваційного підходу до управління кадровим складом, не виключається і використання інших перерахованих вищих типів управління.

За інноваційного управління розвитком персоналу команди розробників програмних систем доцільно використовувати результати теорії управління соціально-економічними системами, отримані у суміжних сферах:

– ефективні механізми стимулювання персоналу команд;

– інноваційні механізми управління розвитком, зокрема функціонуванням динамічними організаційними системами;

– сучасні механізми управління персональним складом фахівців організаційних систем;

– ефективні механізми управління підвищення фахових компетенцій команд розробників програмних систем.

Персонал – це особовий склад групи, працівники підрозділу чи підприємства, яких об’єднують у групи за фаховими або іншими ознаками [262]. Ефективна інноваційна система розвитку персоналу є

сукупністю організаційних структур фірм розробників програмних систем, методів, методик, процесів та ресурсів, які необхідні для успішного виконання актуальних і перспективних завдань; а також для оптимального задоволення запитів фахівців, пов'язаних із самореалізацією та професійною підготовкою і кар'єрним зростанням:

– регулярне навчання, перепідготовка або підвищення кваліфікації персоналу команд розробників програмних систем;

– забезпечення комфортних умов для організації винахідницької та раціоналізаторської роботи під час розробки програмної системи;

– скорочення термінів для професійної адаптації членів команд розробників;

– фахове тестування оцінки професійних навичок кандидатів у члени команд розробників програмних систем на вакантні посади;

– регулярна перевірка кадрів команд розробників програмних систем на фахові знання щодо виробничих завдань;

– розроблення стратегічних планів щодо кар’єрного росту членів команд розробників програмних систем;

– підготовка кадрового персоналу до заміщення вакантних посад у межах фірми розробників програмних систем.

Розглянемо інноваційні підходи до класифікації завдань управління фаховим розвитком членів команд розробників програмних систем (рис. 3.20).

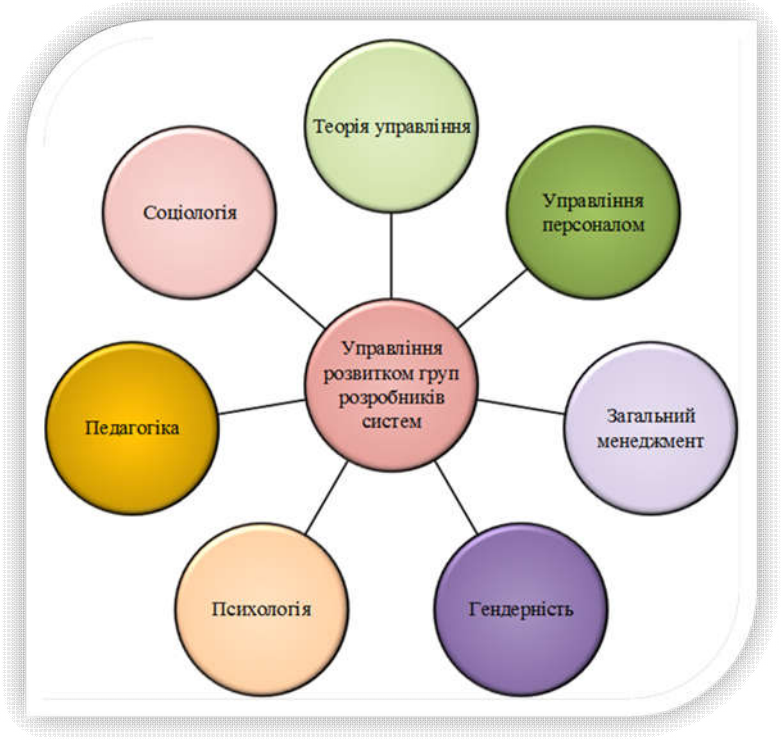


Рисунок 3.20 – Модель управління розвитком групи розробників програмних систем

Інноваційне управління фаховим розвитком групи розробників систем, як комплексний підхід розглядається з погляду різноманітних впливів у дослідженнях:

– теорія управління організаційними системами, а також теорія прийняття рішень;

– управління персоналом (як розділ управлінського менеджменту) та винагорода за якісно виконані завдання;

– управління загальним менеджментом;

– гендерні підходи в управлінні розвитком команд;

– психологічні аспекти при управлінні командою;

– педагогічні аспекти при підвищенні фахових компетенцій членів команд розробників програмних систем;

– соціальні стосунки при управлінні розвитком команди.

На наступному етапі детально проаналізуємо модель класифікації розвитку персоналу з розробки програмних систем (рис. 3.21):

Рисунок 3.21 – Модель класифікації розвитку персоналу з розроблення програмних систем

На першому етапі розглянемо модель розвитку персоналу з погляду особистості – найменшого елемента організації. Особливість цього питання полягає у тому, що організація повинна змінювати з точки зору особистості, щоб ефективність функціонування останньої в даній організації була максимальна або відповідала необхідному рівню.

На другому етапі розглянутої моделі є системи класифікацій задач управління розвитком персоналу (УРП), де число професійних

членів розробників програмних систем фірми, розглядається як певна множина фахових компетенцій, що складається із суми компетенцій кожного окремого члена команди.

На третьому етапі розглянутої моделі є ті підструктури членів команд розробників програмних систем, які є предметом управління фахового розвитку персоналу.

На четвертому етапі розвитку фахових здібностей членів команд розробників програмних систем розглянутої моделі можна виділити (рис. 3.22):

– адаптацію персоналу, яка є неперервним процесом пристосування команди або члена команди до мінливих умов зовнішнього і внутрішнього оточення;

– мотивацію персоналу – створення таких комфортних умов та мотивації до саморозвитку, разом із використанням різноманітних систем мотивації, що стимулюють розкриття особистих можливостей співробітників, їх професійне фахове зростання та саморозвиток. Водночас поза розглядом залишаються складові адаптивної системи мотивації щодо створення належних умов праці та відпочинку персоналу, використання системи соціальних пакетів, різноманітних компенсацій і пільг, що створюють належні умови для збереження та підвищення фахового потенціалу і соціальної захищеності працівників фірми розробників програмних систем.

– забезпечення професійного навчання персоналу (разом із самонавчанням членів команд розробників, підвищенням кваліфікації, перепідготовкою тощо);

– кар’єрне просування членів команд розробників програмних систем (управління кар'єрою разом із плануванням кар'єри, підготовкою резерву тощо).

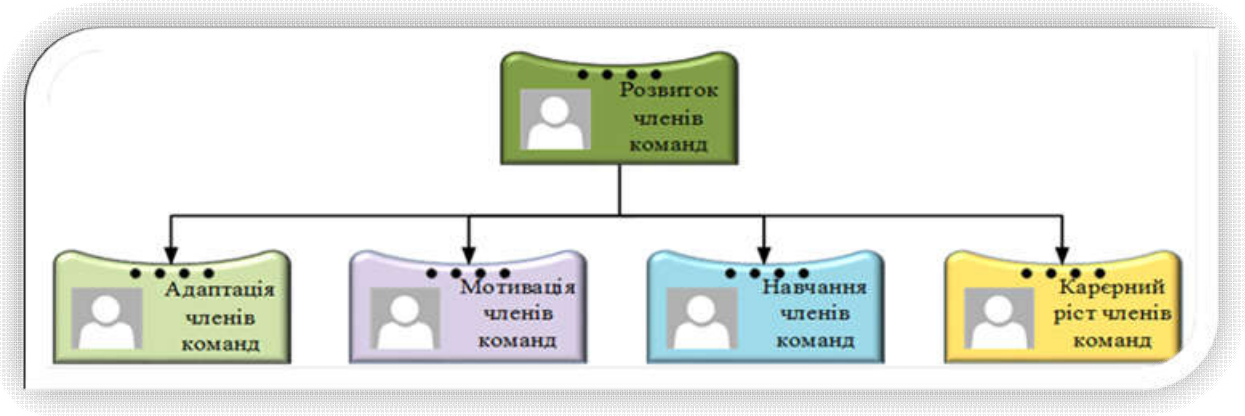


Рисунок 3.22 – Модель розвитку членів команд розробників програмних систем

Отже, отримуємо систему класифікацій задач управління розвитком членів команд розробників програмних систем (табл. 3.4).

Таблиця 3.4 – Система класифікації задач управління розвитком членів команд

<table>
<tr><th colspan="3">УПРАВЛІННЯ ФАХОВИМ РОЗВИТКОМ ЧЛЕНІВ КОМАНД</th></tr>
<tr><td rowspan="2">З погляду зору структури організації (предмет управління «прив'язаний» до функцій членів команд)</td><td colspan="2">З погляду особи (предмет управління з позицій організації «прив'язаний» до ефективності виконання співробітниками своїх функцій)</td></tr>
<tr><td>Індивідуальний розвиток</td><td>Колективний розвиток</td></tr>
<tr><td>Підбір членів команди на вакантні посади
Найм певного члена команди
Функціональні обов’язки
Звільнення члена команди</td><td>Адаптація
Мотивація
Навчання
Кар’єрний ріст</td><td>Адаптація
Мотивація
Навчання</td></tr>
<tr><td colspan="3">Соціальний пакет, компенсації та пільги</td></tr>
</table>

Отже, для побудови моделей індивідуального та колективного розвитку необхідно спочатку сформулювати та дослідити модель ієрархії потреб. Також потрібно зазначити, що наведений комплекс моделей не можна розглядати як вичерпний перелік різноманіття завдань управління розвитком команди розробників програмних систем.

Насамперед демонстрація можливості та доцільності використання в розробці програмних систем з використанням хмарних технологій апарату математичного моделювання, а також ілюстрація можливих методичних підходів до побудови і дослідження моделей розвитку персоналу, які трапляється у різних ситуаціях на практиці (діючи у новій ситуації за певною виробничою аналогією, де можна побудувати адекватну модель і отримати змістовно інтерпретовані висновки) [124].

3.7. Модель ієрархії стимулювання потреб членів команд розробників

Детально проаналізуємо опис формальної моделі, яка аналізує ієрархію потреб будь-якого члена команди розробників програмних систем. Нехай існує *n* упорядкованих потреб, перші *k* з яких є первинними.

Ступінь (рівень) задоволення *i*-тої потреби будемо вимірювати числом $x_i \in [0;1], i \in N = \{1, 2, \ldots, n\}$ – множина потреб. Припустимо, що ступінь задоволення *i*-тої потреби залежить від ресурсу $u_i \geq 0$, що орієнтований на задоволення цієї потреби, і від рівнів задоволення потреб нижніх рівнів:

$$x_1(u_1, u_2, \ldots, u_i) = min\left\{f_i(u_i), \frac{min}{j = 1, i-1} \alpha_{ij} x_j\right\}, i \in N, \qquad (3.37)$$

де $f_i: \Re_+^1 \to [0;1]$ – відомі строго монотонні безперервні функції, $\alpha_{ij} \in [0;1]$ – константи (ваги), що відображають взаємозв'язок між потребами, $j \leq i, i \in N$.

Оскільки практично будь-яку індивідуальну специфіку у потребах можна врахувати за допомогою підбору відповідних функцій $f_i(\cdot)$ та постійних констант $\{\alpha_{ij}\}$, то в якості агрегованої міри задоволення потреб $s \in [0;1]$ виберемо міру задоволення вищою з потреб:

$$s(u) = x_n(u), \text{ де } u = (u_1, u_2, \ldots, u_n) \in \Re_+^n \qquad (3.38)$$

де $s(u)$– вектор ресурсів.

Введемо у розгляд граф *(N, E)*, де безліч дуг *E* є сукупністю дуг від кожної вершини, які відповідають відповідній потребі, до усіх вершинних потреб більш високого рівня. Обчислимо «потенціал» *i*-тої вершини:

$$x_i^{max} = \frac{min}{j < i}\left(x_j^{max} \cdot \alpha_{ij}\right), i \in N \backslash \{1\}. \qquad (3.39)$$

Вирази (3.37) і (3.38) дають змогу за заданих функцій $f_i(\cdot)$ векторі ресурсів *u* знайти міру задоволення потреб. Можна вирішити і зворотне завдання пошуку мінімальних значень ресурсів *u* (s*),* що забез-

печують досягнення заданого рівня $s* \leq x_n^{max}$, де є задоволення потреб.

Позначимо $\alpha = \|\alpha_{ij}\| i,j \in N$ матрицю ваги (де значення ваги α_{ij} – будемо вважати, що воно дорівнює одиниці, $i \in N$, $f_i^{-1} i(\cdot)$ – функцію, зворотною до функції $f_i(\cdot), i \in N, l_{ij} = l_n(1/\alpha_{ij}), L_i$, де є довжина максимального шляху у графі *(N, E)* з вершини *i* у вершину *n* за умови, що довжини дуг рівні $l_{ij}, i \in N$.

Якщо функція $f_i(\cdot)$ набуває значення s* за кінцевих значень ресурсу, то розв'язання цієї задачі, очевидно, виглядає так:

$$u_i^*\left(s^*,\alpha\right) = f_i^{-1}\left(\exp\left(L_i\right)\right), i \in N. \tag{3.40}$$

Отже, мінімальні значення ресурсів, що забезпечують досягнення заданого рівня $s^* \leq x_n^{max}$ задоволення потреб, можна визначити за допомогою виразу (3.40).

Динамічна модель стимулювання членів команд

Припустимо, що первинні потреби не є насиченими, тобто $u_i(t) = q_i, i = \overline{1,k}$, а вторинні потреби – такі, що насичуються, тобто $u_i(t) = q_i\, t, i = \overline{k+1,n}$. Для простого вирішення тут і далі, будемо вважати, що $\alpha_{ij} = 1, i \in N, j \leq i$. Тоді $L_i = 0, i \in N$, і отримуємо такі рівняння динаміки мір задоволення потреб залежно від вектора $q = (q_1, q_2, \dots, q_n)$ ресурсів, які споживаються за одиницю часу:

$$x_i(q_1, q_2, \dots, q_i, t) = \min_{j = \overline{1,i}} = f_j(q_j), i = \overline{1,k}, \tag{3.41}$$

$$x_i(q_1, q_2, \dots, q_i, t) = min\left\{\min_{j = \overline{1,k}} = f_j(q_j), \min_{m = \overline{k+1,i}} f_m(q_m t)\right\}, \tag{3.42}$$

$i = \overline{k+1,n}$.

Вектор ресурсів повинен відповідати збалансованому обмеженню:

$$\sum_{i \in N} q_i \leq Q. \tag{3.43}$$

Отримаємо необхідну умову за якої досягається рівень задоволення потреб *s** за визначений час. Отже, для досягнення агреговано-

го рівня задоволення потреб $s^* \le x_n^{max}$ за кінцевий час достатньо виконати таку умову:

$$\sum_{i=1}^{k} f_i^{-1}(s^*) < Q. \tag{3.44}$$

Ефективність моделей мотивації членів команд

Розглянемо тепер задачу щодо швидкодії задоволення потреб, тобто про мінімізацію часу *T* досягнення заданого рівня $s^* \in [0; 1]$ задоволення потреб шляхом розподілу визначеного ресурсу за його заданих обмежень. Мінімальний час (результат розв'язання задачі) позначимо *T**.

З доведених тверджень випливає основна ідея, яка полягає у тому, що всі вторинні потреби повинні досягати необхідного рівня одночасно. Отже, якщо $s^* \le x_n^{max}$ і виконано умову (3.44), то розв'язання задачі щодо швидкодії задоволення потреб набуде такого вигляду:

$$q_i = f_i^{-1}(s^*), i = \overline{1,k}, \tag{3.45}$$

$$q_m = \frac{f_m^{-1}(s^*)}{\sum_{l=k+1}^{n} f_l^{-1}(s^*)} \left(Q - \sum_{i=1}^{k} f_i^{-1}(s^*) \right), m = \overline{k+1,n}, \tag{3.46}$$

$$t^*(s^*, Q) = \frac{\sum_{l=k+1}^{n} f_l^{-1}(s^*)}{Q - \sum_{i=1}^{k} f_i^{-1}(s^*)}. \tag{3.47}$$

Отримані співвідношення дають також змогу розв'язувати задачу термінального управління, тобто мінімізацію виділених сумарних ресурсів для досягнення певних результатів за заданий час, для забезпечення необхідної міри задоволення потреб чи максимізації агрегованого рівня задоволення потреб за заданий час при фіксованих обмеженнях на ресурси.

З математичних виразів (3.45), (3.47) можна отримати залежність *s*(t),* що описує (при оптимальному розподілі наявного ресурсу) залежність міри задоволення потреб від часу.

Для випадку, коли $\forall i \in N\ f_i(\cdot) = f(\cdot)$, отримуємо:

$$s^*(Q, t) = f\left(\frac{Q\,t}{n - k + kt}\right). \tag{3.48}$$

Величина:

$$k(Q,t) = \frac{s^*(Q,t)}{Q \cdot t} \qquad (3.49)$$

може розглядатися як ефективність використання наявних ресурсів організації для задоволення потреб (мотивацію) членів команд розробників програмних систем.

Припустимо, що функція $f(\cdot)$ має обмежену похідну. Тоді, підставляючи (3.48) в (3.49), обчислюючи похідну за часом, у такий спосіб отримуємо, що справедливою буде така думка: з часом ефективність витрачання наявних ресурсів для мотивації членів команд розробників програмних систем значно зменшується.

3.8. Динамічна модель управління кар'єрою членів команд розробників програмних систем

Кар'єра – це «шлях до успіху, провідного становища у суспільстві, на службовому терені». Типи кар'єри є внутрішньо-організаційний та між організаційний, вертикальний, горизонтальний, ступінчастий, спеціалізований, неспеціалізований. Для підвищення кар'єрного росту члена команди розробників програмних систем розглянемо завдання управління нею з погляду індивіда (індивідуальної кар'єри), а також і з погляду організації (просування персоналу).

Для фіксованого *i*-го члена команди розробників програмних систем введемо орієнтований граф *(V, E)*, де вершини якого відповідають можливим посадам, які він може займати в організації, причому вершини v_{ij} впорядковані у тому сенсі, що дуги йдуть тільки від вершин з меншим першим індексом *к* до вершин з більшим першим індексом. Змістовно, перший індекс $i \in I = \{1,2,\ldots,m\}$ відображає номер рівня ієрархії, а інший індекс ієрархії $j \in J(i)$ – де безлічі посад на *i*-му рівні кар'єрної ієрархії. Довжину дуги $t_{i,j}^{k,i} \geq 0$, $t_{i,j}^{k,i} = +\infty$ при $k < i$ з вершини *i, j* у вершини *k, l* будемо вважати, що показник відображає час, який необхідно пропрацювати на посаді *j* рівня *k*, для того, щоб зайняти посаду *l* на рівні *k*.

Для кожної пари вершин *i, j* та *k, l*, *k*> *i* можна знайти довжину $T_{i,j}^{k,i}$ найкоротшого шляху, що з'єднує ці відповідні вершини:

$$\tau_{i,j}^{k} = \min_{i \in J(k)} T_{i,j}^{k,t}. \quad (3.50)$$

Величина (3.50) може інтерпретуватися як мінімальний час, необхідний для того, щоб, починаючи з *j*-ой посади на *i*-му рівні ієрархії, досягти *k*-го рівня посадової ієрархії. Водночас величина:

$$\tau_{0}^{k} = \min_{i \in J(k)} T_{0}^{k,i} \quad (3.51)$$

відображає мінімальний час, який необхідний для того, щоб, «стартуючи» з самого початку професійної кар'єри, досягти *k*-го рівня ієрархії на підприємстві.

Розглянемо марківський ланцюг, вершини якого відповідають рівням ієрархії посад у розглянутій організації, тобто належать впорядкованій безлічі $I = \{1, 2, \ldots, m\}$. Додамо $m + 1$-шу вершину, яка відображає звільнення із організації, і будемо вважати, що відомі ймовірності переходів, де: p_{ij} – ймовірність того, що і в наступному періоді член команди розробників програмних систем залишиться на тому ж (*i*-му) рівні, p_{ij} – ймовірність того, що він перейде на *j*-й рівень; $j>i$, p_{im+1} – ймовірність того, що звільниться (ймовірність переходу p_{im+1}, $m + 1$, будемо вважати, що дорівнює одиниці. Припустимо, що, один раз звільнившись з даної організації, співробітник у неї більше не повернеться). Ймовірності p_{ij}, $j < i$, величини будемо вважати рівними нулю оскільки, зниження у посаді неможливе. Розраховуючи мінімальні шляхи у графі (рис. 3.23), отримаємо: τ_0^1=1, τ_0^2=2, τ_0^3=3, τ_0^4=4.

На графі зображено чотири рівні ієрархії (m = 4), де числа біля дуг позначають їх довжину, а числа всередині кружечків-вершин – довжину мінімального шляху від входу (нульової вершини) до цієї вершини. Потовщеними лініями зображено найкоротший шлях від входу до найвищого рівня.

Нагадаємо, що величина τ_0^i характеризує запланований *i*-м членом команд розробників програмних систем час досягнення максимального рівня ієрархії, $i = \overline{1,4}$. Отже, у такий спосіб можемо розглянути марківський ланцюг (рис. 3.24).

Нехай значення ймовірностей переходу описуються за допомогою матриці у табл. 3.4.

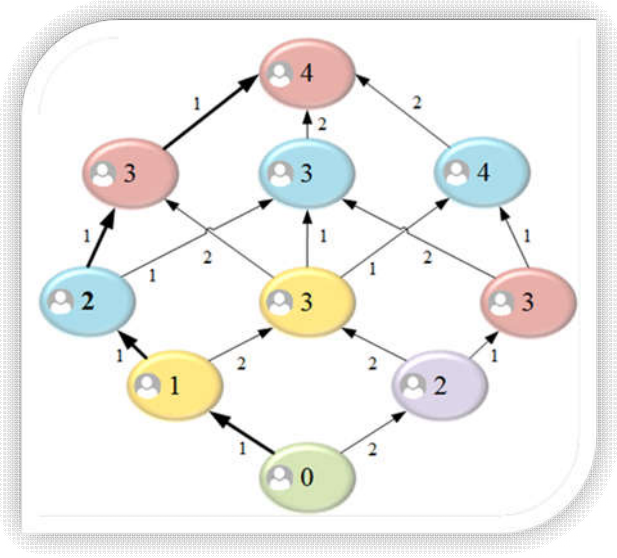


Рисунок 3.23 – Індивідуальна графова модель управління кар'єрою

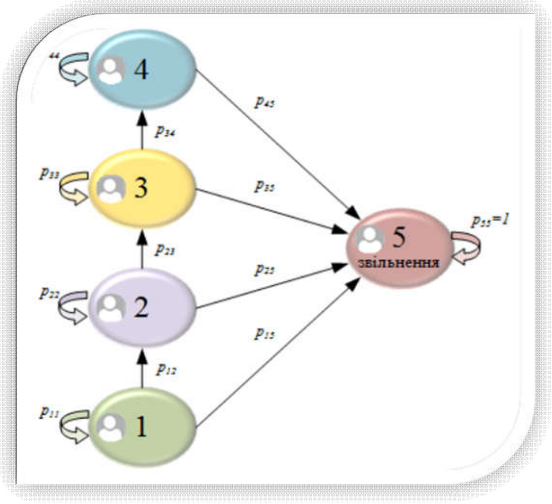


Рисунок 3.24 – Модель марківського ланцюга кар'єрного росту

Таблиця 3.4 – Матриця ймовірностей переходів

	1	2	3	4	5 (Звільнення)
1	0,70	0,20	0,00	0,00	0,10
2	0,00	0,80	0,10	0,00	0,10
3	0,00	0,00	0,70	0,10	0,20
4	0,00	0,00	0,00	0,90	0,10
5 (Звільнення)	0,00	0,00	0,00	0,00	1,0

Позначимо $p(0) = (0,0,\dots,1,\dots,0,0)$, де $m+1$ – мірний стохастичний вектор, всі компоненти якого, крім однієї (не дорівнюють l), а дорівнюють нулю. Це компонента, номер якої відповідає рівню ієрархії l, на якому перебуває або надходить на роботу член команди розробників програмних систем. Матрицю перехідних ймовірностей позначимо P= $||p_{ij}||$.

Тоді динаміка $p(t)$ станів марківського ланцюга буде відповідати умові: $p(t) = p(0)\ \ P^t, t = 1,2,\dots$,

де

$$p(1) = (0.7, 0.2, 0, 0, 0.1),$$

$$p(2) = (0.49, 0.3, 0.02, 0, 0.19),$$

$$p(3) = (0.24, 0.339, 0.065, 0.006, 0.35), \qquad (3.52)$$

$$p(4) = (0.058, 0.22, 0.088, 0.032, 0.602).$$

По суті, $p_i(t)$ – ймовірність того, що у момент часу *t* співробітник буде перебувати на *i*-му рівні ієрархії, $i \in I$. Будемо розглядати *i*-го члена команди розробників програмних систем, що надходить на найнижчий рівень ієрархії в організації. Тоді рішення завдання про індивідуальну кар'єру набуватиме такого вигляду $(\tau_0^k)_{i \in I}$, що є сукупністю мінімальних термінів часу, за який член команди розробників програмних систем планує досягти відповідного рівня ієрархії. Виконання завдання про просування персоналу можна представити у вигляді матриці $p_{ij} = p_j(\tau_0^k)$, $i, j \in I$, рядки якої містять ймовірності того, що у цей момент τ_0^k співробітник буде перебувати на *j*-му рівні ієрархії. Можна вводити різні агреговані критерії узгодженості планів індивіда з пропозиціями кар'єрного росту в організації. Наприклад, ймовірність не успішної кар'єри (з погляду цього працівника) як максимальну ймовірність до того, що рівень ієрархії, на якому буде перебувати певний член команди розробників програмних систем, виявиться значно меншою ніж, та якої він очікував:

$$Q = \max_{i=1,m} \sum_{j<i} p_{ij}\,(\tau_0^k). \qquad (3.53)$$

Із (3.53) отримаємо $Q = \max\{0, 0.49, 0.58, 0.4\} = 0.58$.

Тобто ймовірність неуспішної кар’єри *i*-го члена команди розробників програмних систем, кар’єрні плани якого описуються графом, (рис. 3.23) в організації, які описуються марківським ланцюгом (рис. 3.24) дорівнює 0,58. Це значення досить велике, і навряд чи розглянутий співробітник прийме рішення влаштовуватися на роботу в цю організацію. З погляду управління кар'єрою (якщо точніше, пропозицією кар'єри) в такій ситуації потрібно зменшувати ймовірність неуспішної кар'єри, насамперед, напевно, завдяки зменшенню ймовірностей звільнення з різних рівнів адміністративної ієрархії. Нехай плинність кадрів вдалося знизити, і нові значення ймовірностей переходу описує матриця у (табл. 3.5).

Таблиця 3.5 – Матриця ймовірностей переходів з врахуванням плинності кадрів

	1	2	3	4	5 (Звільнення)
1	0,50	0,49	0,00	0,00	0,01
2	0,00	0,50	0,49	0,00	0,01
3	0,00	0,00	0,50	0,49	0,02
4	0,00	0,00	0,00	0,95	0,05
5 (Звільнення)	0,00	0,00	0,00	0,00	1,00

Тоді по-новому розраховуємо динаміку *p(t)* станів марківського ланцюга, який буде відповідати умові: $p(t) = p(0)\ \ P^t, t = 1,2,\dots$,

де

$$p(1)=(0.5,0.49,0,0,0.01),$$

$$\begin{aligned} p(2) &= (0.25,0.49,0.24,0,0.02), \\ p(3) &= (0.063,0.245,0.36,0.288,0.044), \\ p(4) &= (0.004,0.031,0.105,0.701,0.16). \end{aligned} \qquad (3.54)$$

Із (3.54) отримаємо: $Q = \max\{0,0.25,0.31,0.14\} = 0.31$.

З огляду на керуючі впливи, значення ймовірності для неуспішної кар'єри зменшилася майже вдвічі – до 0,31. Водночас ймовірність звільнення за чотири періоди дорівнює 0,16. Напевно, такі умови кар'єрного росту для багатьох членів команд розробників програмних систем можуть виглядати достатньо привабливо.

Отже, у досліджені завдання управління кар'єрою сформульовано як задача узгодження інтересів членів команд у фірмі. Показано, що взаємовигідні рішення можуть прийматися на підставі порівняння результатів розв’язання задачі планування індивідуальної кар'єри (яка зведена до завдання пошуку найкоротшого шляху у мережі) та завдання просування членів команд у кар’єрі, тобто задачі побудови і дослідження властивостей марківського ланцюга. В нашому дослідженні розглянуто теоретико-ігрові та оптимізаційні моделі управління розвитком членів команд розробників програмних систем у фірмах, тобто вплив на працівників фірми який здійснюється з метою підвищення ефективності їхньої діяльності з погляду інтересів цієї фірми.

Введена система класифікацій задач управління членами команд розробників програмних систем, з погляду фірми виділено такі завдання: підбір членів команд, формування, розподіл функціональних обов'язків та звільнення. З погляду особистості запропоновано розглядати завдання адаптації, мотивації, навчання і просування членів команд.

Висновки до третього розділу

У розділі проведено системний огляд та наведено низку оригінальних результатів, які не є вичерпними у дослідженні математичних моделей формування та функціонування членів команд розробників програмних систем з використанням хмарної технології. Проте вони відображають загальну методологію побудови та вивчення прикладних математичних моделей динамічного функціонування фірм – розробників програмних систем, а також можуть бути ефективно використані при вирішенні задач широкого класу управління соціально-економічними системами.

З погляду теорії дослідження математичних моделей потрібно визнати, що різноманітні результати вивчення команд розробників програмних систем, отримані у психології та соціології, на сьогодні в формальних моделях знаходять недостатньо повне відображення. Для багатьох моделей існують певні труднощі в отриманні аналітичних рішень. Майже не враховується «галузева» специфіка (наприклад, такий поширений на практиці клас формування та існування команд з врахуванням психотипів кожного із членів команди у спортивних командах, не є ще предметом глибокого системного, формального, теоретичного дослідження.

Перспективним напрямом подальших прикладних досліджень, на нашу думку, є розширення класу формування з використанням математичних моделей співіснування психотипів членів реальних організацій і команд, для яких формулюються та використовуються формальні моделі управління.

Отже, у цьому дослідженні розглянуто комплекс механізмів організаційного управління інноваційним розвитком фірми розробників програмних систем, зокрема механізми фінансування, управління організаційними проєктами, інституційного управління, мотивації персоналу та управління розвитком персоналу. Багато класів моделей розглядаються вперше: модель саморозвитку, ігри зі змінним складом,

багатокритеріальна модель стимулювання, моделі ієрархії потреб та управління кар'єрою. Їхній подальший розвиток – це перспективне завдання майбутніх досліджень.

Відзначимо, що основним апаратом дослідження розробки механізмів управління інноваційним розвитком фірми та її особового складу було використано математичне моделювання. З одного боку, їхнє використання математичних моделей дає змогу отримати обґрунтовані висновки, встановити кількісний взаємозв'язок між суттєвими явищами та процесами. З іншого боку, потрібно пам'ятати, що, будуючи будь-яку математичну модель, вводять низку припущень і результати аналізу цих моделей справедливі тільки в їхніх межах.

Тому, напевно, не варто розглядати результати математичного моделювання як остаточний результат, а як деякий «алгоритм», підставивши в який числові значення, що відповідають тій чи іншій реальній ситуації, можна отримати вичерпну відповідь на питання, як управляти інноваційним розвитком команд розробників програмних систем та фірм, у яких вони існують. Переваги математичного моделювання полягають у тому, що воно дає змогу: пояснювати спостережувані явища і зв'язки між ними; прогнозувати майбутній розвиток процесів взаємовідносин між членами команди розробників програмних систем, тобто вибирати оптимальні варіанти розвитку.

РОЗДІЛ 4

КОНЦЕПЦІЯ РОЗРОБЛЕННЯ РОЗПОДІЛЕНИХ ПРОГРАМНИХ СИСТЕМ З ВИКОРИСТАННЯМ ХМАРНОЇ ТЕХНОЛОГІЇ

4.1. Базові підходи, моделі та принципи розроблення розподілених програмних систем

Сьогодні програмні системи на основі хмарної технології стають дедалі популярнішими. Багато процесів, які донедавна виконувалися на персональних комп'ютерах, зараз проводять за допомогою Web-оглядача та забезпечення доступу до різних онлайн сервісів. Потреба у розподілених програмних системах на основі хмарної технології за останні десять років виросла настільки, що обчислювальних ресурсів одного серверу необхідно дедалі більше. Інтернет, а заодно і розподілені програмні системи проникли у всі сфері діяльності людини. Розподілені Web-сервіси чи програмні додатки можна побачити в енергетиці, медицині, освіті, машинобудуванні та інших. Поступово індустріальне суспільство відходить від класичного використання програмних систем, що можна виконувати тільки на обмежених архітектурних рішеннях та наявній мінімальні обчислювальній продуктивності. Мобільність програмних систем – це основна потреба більшості користувачів інформаційного суспільства [182]. Розподілені програмні системи з використанням хмарної технології на незалежних обчислювальних платформах дають змогу працювати на будь-яких сучасних інформаційних пристроях із будь-якого віддаленого місця, а отже, вони позбавлені класичного недоліку [8].

Такому динамічному розвитку розподілених програмних систем на основі хмарної технології сприяло тотальне використання процесів у сфері розробки багатоядерних центральних процесорів, а також і стрімке поширення надшвидкісних мереж із доступом до Інтернету. Проте у сучасному світі потреба в обчисленнях невпинно зростає, отже, продуктивності обчислювальних систем потрібно дедалі більше. Доповнює сучасний тренд використання інформаційних технологій стрімке зростання кількості потенційних користувачів розподілених програмних систем на базі хмарної технології. На тлі цієї проблеми в сфері обчислю-

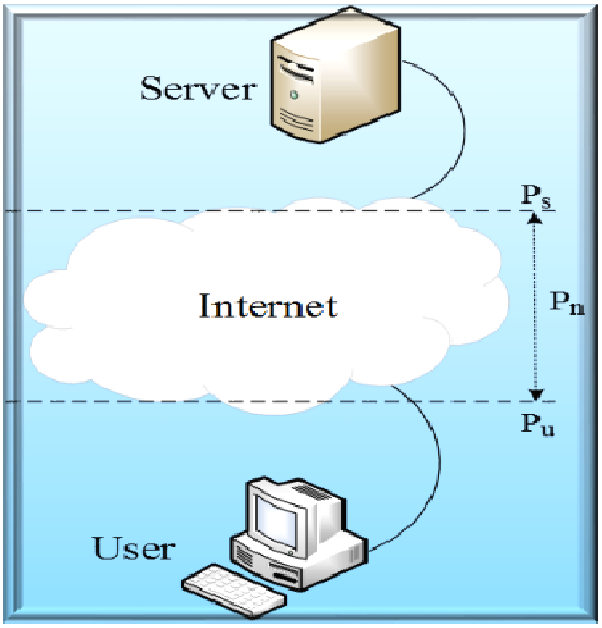
вальних ресурсів з'явилася нагальна потреба у розробленні високонавантажуваль-них розподілених програмних систем.

Розподілена програмна система на базі хмарної технології – це набір розподілених обчислювальних вузлів, які об'єднані у єдину структуру, що працює як один механізм. Головною її перевагою є те, що вона може легко збільшувати обчислювальні ресурси завдяки простому додаванню нових вузлів. Розробка високонавантажувальних відмовостійких програмних систем – це базова властивість будь-якої системи, яка полягає у тому, щоб забезпечити розробленій програмній системі на базі хмарної технології подальше функціонування, якщо виникли нештатні помилки (збурення) у одній або декількох її частинах [192-194]. Створюючи високонавантажувальні відмовостійкі розподілені програмні системи, зазвичай віддають перевагу такій якості системи, як відмовостійкість на відміну від швидкодії обчислювальних ресурсів. Головною причиною вибору цього критерію є те, що тимчасове часткове зниження продуктивності програмної системи на базі хмарної технології більш допустиме для багатьох користувачів, аніж відсутність доступу загалом та втрати інформаційних потоків даних упродовж нетривалого часу.

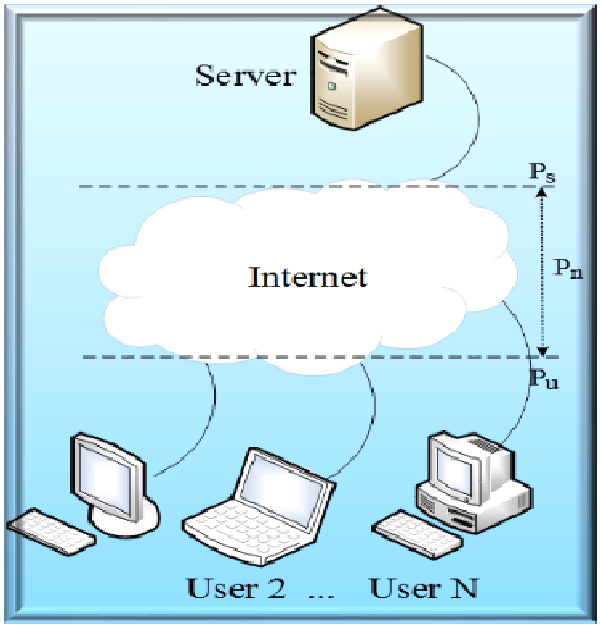
Розглянемо архітектурні рішення щодо розроблення розподілених відмовостійких програмних систем на основі хмарної технології (рис. 4.1).

На початковому рівні це лише сполучення користувача з віддаленим ресурсом через Інтернет (рис. 4.1a). На другому рівні розроблення розподілених програмних систем забезпечується багатокористувацьке під'єднання до одного сервера (рис. 4.1b). На третьому рівні багатокористувацьке під'єднання до різноманітних серверів (рис. 4.1c) і на четвертому рівні при багато користувацькому під'єднані сервери об'єднуються в один обчислювальний кластер, водночас забезпечують об'єднання обчислювальних можливостей вузла (рис. 4.1d).

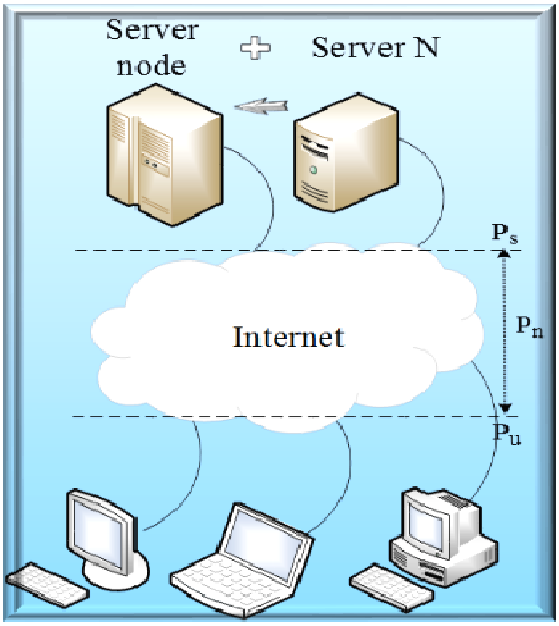
Однак масштабованість розроблення високонавантажувальної відмовостійкої розподіленої програмної системи на основі хмарної технології полягає у тому, що інформаційні ресурси або доступ до них розподіляються між декількома обчислювальними серверами (вузлами) [5, 10]. У дослідженні наведено деякі базові принципи розроблення програмних систем з використанням хмарної технології на незалежних обчислювальних платформах, які здійснюють значний вплив на ідеологію та дизайн великих розподілених програмних систем.



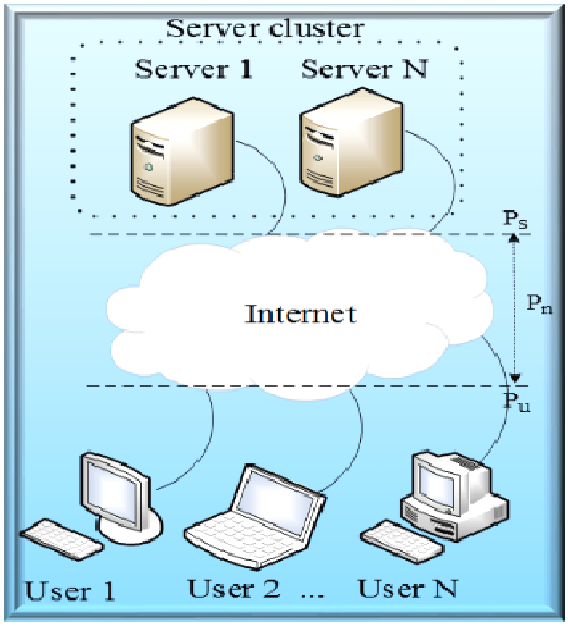
a) Базова розподілена програмна система



b) Розподілена багатокористувацька програмна система



c) Багатокористувацька, багатосерверна розподілена програмна система



d) Багатокористувацька кластерна розподілена програмна система

P_n – переміщення навантаження на сервер або до користувача (тонкий або товстий клієнт); P_s – навантаження на сервер або кластер; P_u – навантаження на комп'ютер користувача

Рисунок 4.1 – Моделі архітектур розподілених програмних систем з регулюванням обчислювального навантаження

Важливими критеріями у розробленні програмних систем є:

Доступність – це вкрай важливий критерій для успішної реалізації бізнес-логіки багатьох компаній. Широка доступність у розподілених відмовостійких програмних системах на основі хмарної технології вимагає детальної розробки надлишкової функціональності бізнес-систем для її ключових компонентів, а також швидкого відновлення працездатності, якщо виникають часткові несправності, та керованості у різних проблемах.

Продуктивність – це ефективне використання обчислювальних ресурсів, що є наріжним каменем для більшості високонавантажувальних розподілених інформаційних систем. Швидкість роботи програмної системи на основі хмарної технології суттєво впливає на задоволення користувачів, а також на рейтингове місце у пошукових системах та створює ключовий фактор, який тісно пов'язаний із доходом комерційних компаній. Отже, отриманий результат є головним у розробці розподілених програмних систем, який оптимізований для оперативного реагування при виникненні виробничих проблем і забезпечує невеликі затримки в обчислювальному процесі опрацюванні інформаційних даних.

Надійність – це відмовостійкість системи, яка повинна бути достатньо стійкою, щоб інформаційні запити щодо даних були правильно опрацьовані та забезпечили їхню достовірність у потрібний час та необхідному вигляді. Інформаційний запит до хмарного сховища даних повинен надавати оновлені опрацьовані дані тоді коли вони змінюються або модифікуються. Користувачам розробленої високонавантажувальної розподіленої програмної системи на основі хмарної технології необхідно гарантувати надійність зберігання й опрацювання даних, їхню достовірність та доступність.

Масштабованість – це така характеристика високонавантажувальної розподіленої інформаційної системи, коли йдеться про будь-яку ґрунтовну систему, де її розмір є лише єдиним, проте головний аспект масштабу, який розглядається насамперед під час розроблення. Не менш факторами аспектами розроблення програмних систем є критерії, які необхідні для забезпечення бажаної продуктивності при опрацюванні значної кількості інформаційних потоків даних. Масштабованість розподіленої програмної системи на основі хмарної технології може стосуватися різних аспектів: яку кількість додаткового інформаційного трафіку вона може опрацювати за визначений час, чи достатньо легко можна збільшити об’єм оперативної пам’яті, а як на-

слідок чи є можливість значно покращити обчислювальну потужність при опрацюванні транзакцій.

Керованість – це така інтегральна характеристика при розробленні програмних системи з використанням хмарної технології на незалежних обчислювальних платформах, яка забезпечує зручне керування обчислювальними ресурсами і є вкрай важливим моментом у функціонуванні її. Управління розподілених програмних систем на основі хмарної технології прирівнюються до масштабованості обчислювальних операцій, а саме обслуговування та оновлення. Реалізація керованості програмної системи забезпечує легку діагностику та розуміння проблем функціонування тоді, коли вони трапляються, а як результат керованості легкість внесення виправлень у програмний код системи тим самим забезпечивши її оновлення або модифікацію.

Вартість – це основний та вкрай важливий критерій функціонування високонавантажувальних розподілених програмних систем на основі хмарної технології, який може охоплювати витрати як на апаратне так і на програмне забезпечення, а також необхідно не забувати про інші аспекти життєвого циклу функціонування програмної системи цілому, які необхідні для розгортання та підтримки. З огляду на вартість розроблення та встановлення програмної системи потрібно розглянути всі етапи загалом або окремі модулі окремо. Прикладом таких витрат є час, витрачений на розроблення програмної системи, кількість необхідних оперативних ресурсів та зусиль, які потрібні для управління, і навіть обсяг необхідної фахової підготовки відповідних членів команд для забезпечення певної компетентності. Вартість – це загальний обсяг вартості розробки, впровадження, підтримки і функціонування розробленої високонавантажувальної розподіленої програмної системи на основі хмарної технології.

Кожен із цих розглянутих принципів є вкрай важливим для прийняття правильних управлінських рішень при розробленні розподіленої програмної системи. Проте одночасне застосування різних принципів також може бути достатньо суперечливим щодо один одного. Особливе акцентування до використання одного із критеріїв може значно збільшити загальну сумарну вартість розроблення програмної системи [11]. Наприклад вибір способу розв’язання проблеми недостатності обчислювальних ресурсів методом простого додавання додаткових вузлів таких, як серверів, водночас оптимізує масштабованість функціонування програмної системи та суттєво ускладнює управління додатко-

вими серверами. Тому і збільшиться сумарна вартість їхнього обслуговування.

Отже, при розробці розподіленої програмної системи на основі хмарної технології вкрай важливо розглянути, які критерії й основні принципи у комплексі вирішення поставленої проблеми. Архітектура високонава-нтажувальної розподіленої програмної системи на основі хмарної технології повинна мати певну надлишковість у своїх функціональних можливостях для опрацювання інформаційних потоків даних. Наприклад, якщо на одному обчислювальному сервері є лише одна копія файла, то втрата працездатності (вихід із ладу) цього сервера буде автоматично означати і втрату цього інформаційного файла з даними. Загальним способом уникнення таких проблем є метод створення декількох інформаційних копій цього файла на інших обчислювальних вузлах (серверах). Цей же підхід використовується і для формування інформаційних запитів до хмарних сховищ даних [14, 15, 21, 37, 38].

Якщо для розподіленої програмної системи на основі хмарної технології основним критерієм є функціональність, то забезпечення одночасної синхронізації декількох копій системи або ж її версій може забезпечити захист нашій програмній системі від короткотермінового збою одного з обчислювальних вузлів [33, 53]. Наприклад, якщо у технологічному процесі бізнес-логіки опрацювання інформаційних даних є два екземпляри тієї самої служби і одна з них не працює або видає гірший від початку результат, то система може перейти до робочої копії на іншому обчислювальному ресурсі. Збій розподіленої програмної системи на основі хмарної технології може відбутися будь-коли, а її відновлення може бути як автоматичне, так і виконуватись власноруч [47, 52].

Ще одним дуже важливим критерієм відмовостійкості високонавантажувальних розподілених програмних систем є створення архітектури із автоматичним резервуванням критичних інформаційних масивів даних, де кожен із обчислювальних вузлів здатний виконувати певну функціональність системи незалежно один від одного. Така архітектура розподіленої програмної системи не має центрального обчислювального вузла управління станами та координації дії для інших обчислювальних вузлів. Цей підхід до розроблення програмної системи забезпечує оперативну масштабованість, оскільки нові обчислювальні вузли можна збільшувати без спеціальних перед умов [55, 66, 67]. Отже, використання таких архітектурних підходів до розроблення розподілених програмних систем на основі хмарної технології забезпечує достатню стійкість системи, оскільки у ній немає єдиної точки відмови.

Сьогодні дедалі частіше доводиться оперувати великими наборами інформаційних потоків даних, а для прискорення опрацювання застосовуються алгоритми поширення певної частини цих даних у різні обчислювальні сервери. Також дуже часто виникає нагальна потреба, коли обчислювальним алгоритмам необхідна додаткова потужність серверів, і для розв'язання цієї проблеми вони об'єднуються у кластери [62]. Отже, якщо недостатньо обчислювальних ресурсів, можна використати вертикальне або горизонтальне масштабування розподіленої відмовостійкої програмної системи на основі хмарної технології.

Якщо архітектор при розробленні програмної системи вирішить застосувати вертикальне масштабування, то це буде означати додавання додаткових обчислювальних ресурсів при використанні окремого сервера [73]. Відповідно до цього, стає зрозуміло, що для опрацювання багатьох обчислювальних даних може знадобитися більше жорстких дисків водночас забезпечуючи можливість опрацювання інформаційних потоків даних на одному сервері. Якщо для обчислення потрібно багато апаратних ресурсів, то їх викладають на потужному сервері, де є більше обчислювальних ядр, частоти, оперативної пам'яті тощо. У кожному окремому випадку при розробленні програмних систем на основі хмарної технології вертикальне масштабування відбувається шляхом створення нового індивідуального ресурсу, який здатний самостійно опрацьовувати поставлені перед ним проблеми та завдання [174, 202, 205, 221].

На відміну від вертикального, горизонтальне масштабування – це додавання у програмну систему на основі хмарної технології інших обчислювальних вузлів. Для опрацювання великих інформаційних потоків даних архітектор програмної системи може запропонувати фізичне додавання сервера для зберігання й обробки частини набору даних, а на рівні обчислювального ресурсу це буде означити розподілення обчислень за різними серверами або ж завантаження та опрацювання між декількома вузлами [197]. Щоб повністю використати переваги горизонтального масштабування програмної системи на основі хмарної технології, її потрібно впровадити ще на стадії розроблення внутрішнього дизайну архітектури системи. Інакше цей підхід до розподілу контенту між обчислювальними вузлами буде дуже складно реалізувати на активній програмній системі [87]. Коли архітектори розуміють, що потрібно застосувати горизонтальне масштабування, то одним із найпоширеніших підходів та методів є розділення обчислювальних можливостей системи [91]. Його можна виконати у такий спосіб, що будь-

який логічний набір обчислювальних функціональних можливостей має бути окремим. Таке архітектурне рішення при розробленні програмних систем може втілюватися завдяки розподілу окремих обчислювальних вузлів за географічним або іншим критерієм [81, 93, 94].

Звичайно, сьогодні все ще існують проблеми із розподіленням обчислювальних даних або їхніх функціональних можливостей між декількома серверами. Одним з основних критеріїв опрацювання інформаційних потоків даних є їхнє розташування. Чим ближче ці дані до проведення операції або до точки обчислення, тим краща продуктивність системи загалом. Отже, при розробленні програмних систем потенційною проблемою є розділення інформаційних потоків даних на кілька серверів, тому що будь-коли інформаційні потоки даних можуть надходити через мережу до обчислювального вузла [96-98, 103-105]. Окрім того, можуть виникнути і непослідовності, якщо існують інші сервісні служби, такі як читання або запису зі спільного інформаційного ресурсу (сховища або бази даних).

Існує критична можливість для виникнення таких умов, як «гонки», де якась частина інформаційних потоків даних до опрацювання повинна бути оновлена, проте зчитування відбувається ще до їх оновлення. Отже, результат опрацювання інформаційних даних не є достовірним і придатним для подальших обчислювальних операцій.

4.2. Інноваційні підходи до побудови розподілених відмовостійких кластерних систем

В умовах тотальної інформатизації сучасного суспільства існує безліч варіантів високонавантажувальних розподілених відмовостійких програмних систем на основі хмарної технології, зокрема: кластери, мейнфрейми, GRID-системи [266]. Проте незважаючи на таке різноманіття методик, підходів та практичних втілень у проєктуванні розподілених програмних систем, не всі відповідають високому рівню необхідної та достатньої відмовостійкості [267]. Окреслимо основні вимоги до розробки високонавантажувальних розподілених відмовостійких програмних систем на основі хмарної технології:

- збільшення продуктивності програмних систем і сервісів завдяки ефективному використанню обчислювальних ресурсів на незалежних обчислювальних платформах та загального збільшення обчислювальних потужностей системи загалом;

- забезпечення працездатності програмної системи на відмовостійкість і доступності до інформаційних потоків даних у режимі 24/7;

- здійснення автоматизованого централізованого резервування інформаційних потоків даних та їх сервісів;

- забезпечення гнучкого управління інформаційними ресурсами при значній зміні навантаження на обчислювальні вузли;

- забезпечення можливості збільшення продуктивності програмної системи на основі хмарної технології без зміни її загальної архітектури;

- мінімізація обсягу трудовитрат та загальної вартості експлуатації програмної системи;

- міграція розробленої програмної системи, сервісів та інформаційних потоків даних між різними обчислювальними вузлами або незалежними платформами.

Змішані кластери у розробці програмних систем

Змішані кластери – це така архітектура високонавантажувальної розподіленої відмовостійкої програмної системи, коли зберігання та опрацювання значних потоків даних відбувається, як на об'єднаних так і окремих обчислювальних серверах. За будовою та проєктуванням вона змішана і називається кластерні системи [20, 63, 78, 110]. Принцип їхньої дії ґрунтується на розподілі інформаційних потоків запитів через один або декілька вхідних обчислювальних вузлів, які свого часу перенаправляють значні потоки даних на опрацювання різними обчислювальними вузлами на незалежних платформах, відповідно до рівня їхнього максимального навантаження [135, 144]. У таких архітектурних рішеннях при проектуванні високонавантажувальних розподілених відмовостійких програмних систем досягається висока надійність системи у цілому без втрат її загальної продуктивності [145, 155, 167, 168]. З погляду користувача вся розроблена програмна система на основі хмарної технології становлять єдину програмну систему. Незалежні обчислювальні вузли, які входять до складу цієї системи, працюють так само, як окремі сервери, проте мають можливість оперативно й автоматично забезпечити балансування загального обсягу навантаження та передавання управління, якщо певний модуль програмної системи перестає працювати.

Програмні системи побудовані на основі кластерів з такою архітектурою, що поєднують у собі різноманітні характеристики високопродуктивних кластерів та кластерів нагальної готовності, але водночас у неї є і недоліки. На відміну від високопродуктивних програмних систем, які орієнтовані на обчислювальний характер певних інформацій-

них потоків завдань, системи із змішаною архітектурою забезпечують високу швидкодію при виконанні операцій введення або виведення, що є у край критичним показником для оцінки швидкодії роботи розробленої розподіленої відмовостійкої програмної системи на основі хмарної технології та сервісів, які обслуговують цю систему [201-203, 210]. Визначення продуктивності обчислювальних операцій вводу або виводу у розробленій програмній системі дає змогу визначити максимальну кількість одночасно працюючих користувачів та інтегрований час реакції програмної системи на різноманітні запити користувачів.

Багато програмних модулів кластерів «гарячої» готовності перебуває у режимі активного очікування опрацювання інформаційних потоків даних на незалежних обчислювальних платформах: якщо будь-який із робочих кластерів виходить із ладу, його замінюють іншим кластером, які перебувають в оперативному резерві [33, 41, 53, 56, 82] . Отже, за такої архітектури проєктування програмних систем неможливо використати кластери, які перебувають у режимі очікування для вирішення інших нагальних завдань, тому обчислювальна ефективність розробленої програмної системи на базі кластерних підходів має незначну продуктивність, а кластери змішаного типу архітектури позбавлені цього недоліку.

Зазвичай типова архітектура змішаних кластерних систем складається із трьох обчислювальних вузлів (серверів), які взаємодіють між собою на рівні розділення інформаційних та апаратних ресурсів. Основними елементами, які розділяють у таких обчислювальних системах, є накопичувачі, які утворюють дискові інформаційні масиви [204, 206, 207]. На одному із обчислювальних вузлів розробленої програмної системи на основі хмарної технології встановлюється спеціальна програма або сервіс – балансувальник загального навантаження, який забезпечує в автоматичному режимі перерозподіл навантаження між цими обчислювальними вузлами кластера. У цьому випадку управління кластером відбувається за допомогою незалежних хмарних сервісів службових повідомлень, що передають інформацію про стан обчислювального вузла. Ця кластерна архітектура має такі переваги:

- можливість рівномірного навантаження, яка забезпечується визначенням загальних правил та підходів при балансуванні обчислювальних запитів між вузлами кластера;

- забезпечення надлишкової відмовостійкості програмної системи за рахунок одночасної роботи декількох обчислювальних вузлів та ре-

зервних каналів зберігання й опрацювання інформаційних потоків даних;

- просте масштабування розробленої програмної системи завдяки збільшенню обчислювальної потужності кластера методом простого додавання певної кількості вузлів;

- можливість гарячого обслуговування та заміни обчислювальних вузлів кластерної системи без зупинки й окремих хмарних сервісів.

Архітектура змішаних кластерів у розробленні програмних систем

Розроблення програмних системи на основі хмарної технології з використанням кластерів змішаного типу можна розділити на однорідні (усі обчислювальні вузли кластера мають однотипну архітектуру і потужність) та гетерогенні, які використовують незалежні обчислювальні вузли із різними архітектурами й потужністю. Зазвичай, коли йдеться про розробку програмних систем на основі хмарної технології з використанням високонавантажувальних обчислювальних кластерів, то архітектори переважно мають на увазі однорідні кластери. Отже, збільшуючи кількість кластерів, потрібно використовувати обчислювальні сервери на незалежних обчислювальних платформах, які відрізняються не тільки за потужністю, але й за своєю архітектурою. У такому випадку однорідний обчислювальний кластер може змінювати свою структуру та переходити у статус неоднорідної структури кластера [134, 146, 147]. При такій архітектурі розроблення програмних систем з використанням неоднорідної структури кластерних систем включають такі проблеми:

- різноманіття у обчислювальних потужностях вузлів значно утруднює завдання розподілу обчислювальних робіт між роботою процесорів;

- різноманіття в архітектурі побудови кластерних систем потребує значної підготовки спеціальних виконавчих файлів для адміністрування окремих вузлів.

При розробці розподілених програмних систем на основі хмарної технології з використанням кластерів та обчислювальних ресурсів (оперативної пам’яті) можна поділити на кластери із загальною пам'яттю, розподіленою пам'яттю (UMA-системи), кластери із фізично розподіленою та загальнодоступною пам'яттю (гібридні системи, NUMA-системи).

Архітектура програмних систем з використанням хмарної технології на базі кластерів із загальною оперативною пам'яттю зазвичай має високу пропускну здатність при передачі інформаційних потоків даних

між процесорами обчислювальної платформи, але за умови, що не відбувається одночасне звернення до кількох процесорів одного і того ж елементу пам'яті. Для кластерів з розподіленою архітектурою пам'яті характерна наявність значної кількості швидких каналів обміну інформаційних потоків даних, які пов'язують окремі частини цієї оперативної пам'яті з окремими процесорами на незалежній обчислювальній платформі. Розроблена програмна система на базі кластера з гібридною пам'яттю має оперативну пам'ять, яка фізично розподілена у різних частинах цієї системи, але логічно розділена та утворює єдиний обчислювальний адресний простір [183]. Вона на незалежних обчислювальних платформах називається загальною логічною пам'яттю (logically shared memory).

Кластери з розподіленими ресурсами складаються із загальної системи зберігання даних та вузлів кластера, які розподіляють доступ до загальних інформаційних потоків даних. Архітектура із загальними дисками більш ефективна за високої потужності системи зберігання інформаційних потоків даних та при опрацюванні завдань [301, 319]. Тому не потрібно тримати кілька копій інформаційних даних і водночас, якщо вузол виходить із ладу, завдання можуть бути миттєво доступні для інших обчислювальних вузлів на незалежних платформах. Відповідно до тверджень, зростання обчислювальної потужності програмної системи можна забезпечити завдяки застосуванню горизонтального масштабування кластера нарощування загальної потужності розподіленої кластерної системи через додавання нових вузлів [201-203, 210]. Та вертикального масштабування, де існує змога нарощування обчислювальної потужності системи завдяки модернізації наявних обчислювальних вузлів кластерної системи або здійснення масштабування, комбінуючи обидва варіанти.

4.3. Кешування та балансування навантаження інформаційних потоків даних програмної системи

Лавиноподібне зростання кількості інформаційних потоків даних та запитів від користувачів спричиняє вирішення двох головних проблем функціонування високонавантажувальних відмовостійких програмних систем масштабування доступу до сервера та баз або банків даних. Методика кешування інформаційних потоків даних ґрунтується на принципі: «нещодавно запитуваних даних, які з великою ймовірніс-

тю будуть запитані знову і знову». Кешування використовуються практично на всіх рівнях опрацювання обчислень як апаратними засобами так і операційними системами, а також за допомогою Web-браузерів, Web-додатків тощо [230, 288-290]. Кешування – це тимчасове збереження інформаційних даних в оперативній пам'яті, проте воно має обмеження. Водночас повторне видобування даних набагато швидше та містить найновіші елементи для повторного опрацювання масивів даних. Кешування може існувати на всіх архітектурних рівнях розробленої програмної системи, але воно найчастіше трапляється на найближчому до опрацювання рівні, де реалізовані інформаційні процеси швидкого повернення масивів даних, без використання інших рівнів.

Розміщення інформаційного кешу безпосередньо на рівні опрацювання запиту дає змогу локально зберігати ці відповіді дані. Щоразу, коли інформаційний запит надається певній службовій програмі, то обчислювальний вузол швидко знайде та поверне локальні кешовані дані, якщо вони існують (рис. 4.2).

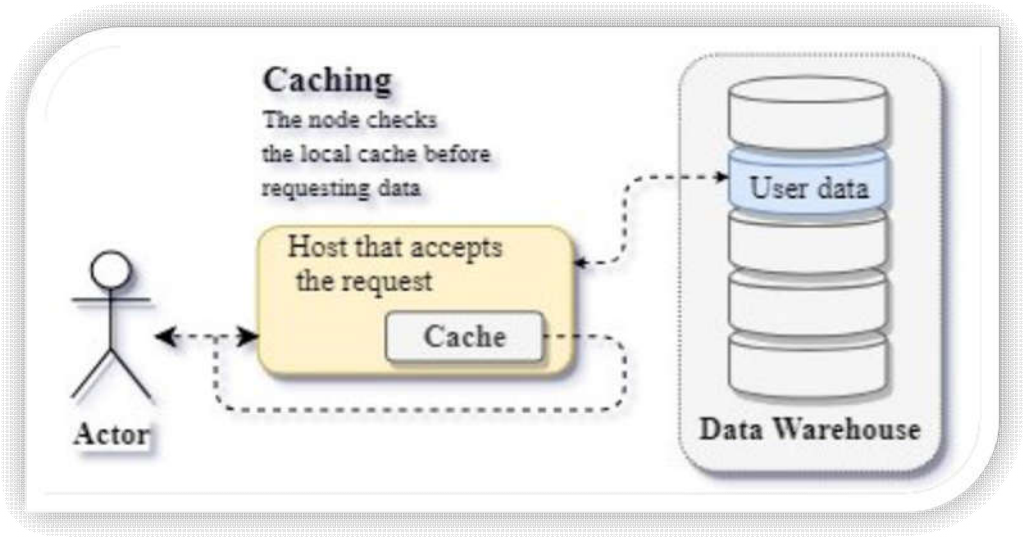


Рисунок 4.2 – Кешування інформаційних потоків даних на рівні запиту користувачів

Якщо інформаційного запиту немає у кеш-пам'яті, то обчислювальний вузол запиту шукає дані на жорсткому диску. Кеш-пам'ять може бути як в оперативній пам'яті (що дуже швидко опрацьовується), так і на локальному диску обчислювального вузла (що є значно швидше, ніж перехід на мережеве сховище даних). Проте якщо розширити цей принцип для декількох обчислювальних вузлів на незалежних обчислювальних платформах, то може виникнути непередбачувана ситуація, коли кожен вузол буде мати свій власний кеш інформаційних даних. Можуть з’явитися нетипові помилки опрацювання кешу, коли балан-

сувальник інформаційного навантаження випадково направлять запити даних від одного користувача в різні вузли. Існує два варіанти вирішення цієї проблемної задачі – організація глобальних або розподілених кешів.

Глобальний інформаційний кеш працює за принципом «один за всіх», тобто усі обчислювальні вузли використовують один і той самий простір кешу. Такий підхід передбачає додавання обчислювального сервера чи файлового сховища інформаційних даних певного типу, яке опрацьовується набагато швидше, ніж оригінальний масив даних. Кожен обчислювальний зовнішній вузол здатний звертатися щодо кешу даних так само, як і локальний. Такий архітектурний підхід до проблем кешування може бути дещо ускладненим. Однак він надзвичайно ефективний разом зі спеціалізованим програмно-апаратним забезпеченням. Існує дві загальні форми організації глобальних інформаційних кешів: коли кешована відповідь не знайдена, тоді сам кеш стає відповідальним за отримання фрагмента даних з бази та банку даних, якого немає (рис. 4.3).

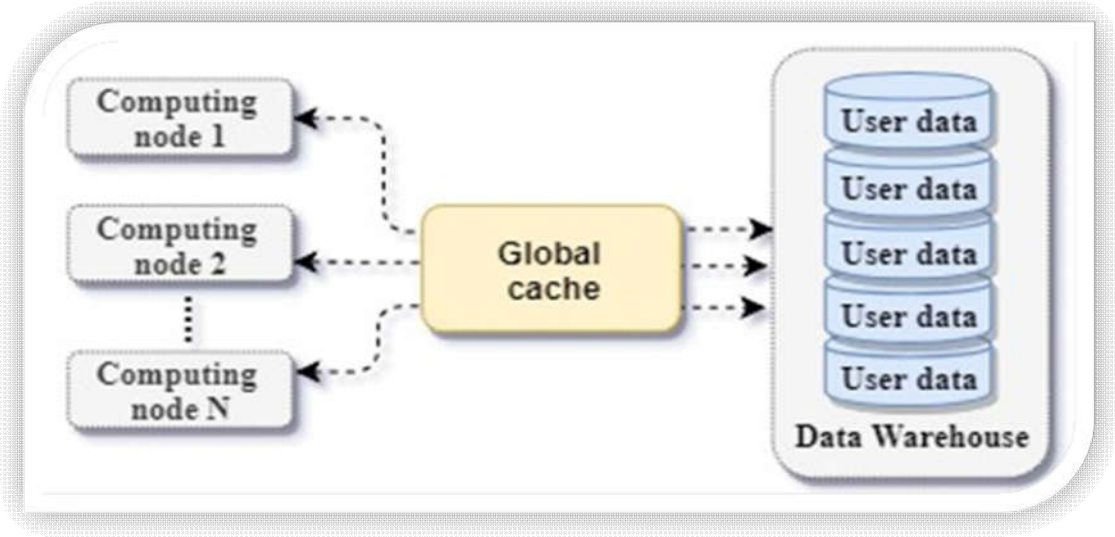


Рисунок 4.3 – Глобальний кеш, який відповідає за отримання інформаційних потоків даних

Проте при опрацювання інформаційних потоків даних на незалежних обчислювальних платформах може виникати проблемна ситуація, коли вузли самостійно здійснюють запит до потоків даних, якщо їх не знайшли в глобальному кеші (рис. 4.4).

Більшість розроблених програмних систем, які використовують глобальні кеші, зазвичай застосовують перший тип, де сам інформаційний кеш керує отриманням потоків даних, щоб запобігти дублюванню запитів на ті самі дані від користувачів. Проте такий підхід мо-

же стати для розробників програмних систем проблемним місцем у ситуаціях непередбачуваного лавино подібного зростання інформаційного навантаження, що є вкрай негативним фактором та може спричинити неконтрольований ланцюг відмов в опрацюванні запитів користувачів у кластерах або обчислювальних вузлах. Тому ще на стадії розроблення програмних систем з використанням хмарної технології необхідно детально оцінювати усі можливості системи перед тим, як запроваджувати алгоритми використання глобального кешу першого типу.

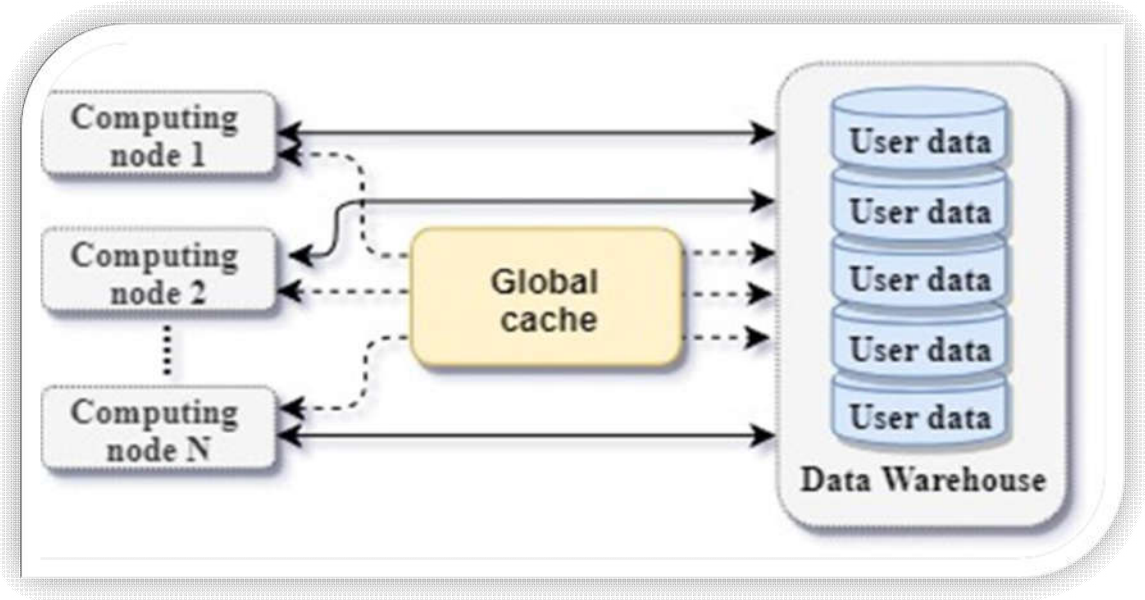


Рисунок 4.4 – Опрацювання потоків даних, де вузли самостійно здійснюють користувацький запит

Аналізуючи сучасні тренди в ефективному використанні інформаційних кешів, зауважимо, що сьогодні дуже активно застосовується організація розподіленого кешу, де кожен із його вузлів має частину кешованих потоків даних. Зазвичай при розроблені програмних систем кеш розподіляється між обчислювальними вузлами на незалежних платформах за допомогою послідовної функції кешування. Коли обчислювальний вузол шукає певний фрагмент потоків даних, то він може швидко отримати інформацію, де знайти його у розподіленому кеші, щоб визначити, чи ці дані доступні [120]. У такому випадку кожен обчислювальний вузол має невеликий масив інформаційного кешу, після чого він надсилатиме користувацький запит на інший обчислювальний вузол перед тим, як перейти до початкового стану опрацювання даних. Тому однією із основних переваг використання розподіленої кеш-пам'яті у розробленні програмних системи є значний обсяг кеш-пам'яті,

який користувач може отримати, просто додаючи обчислювальні вузли до запитів (рис. 4.5).

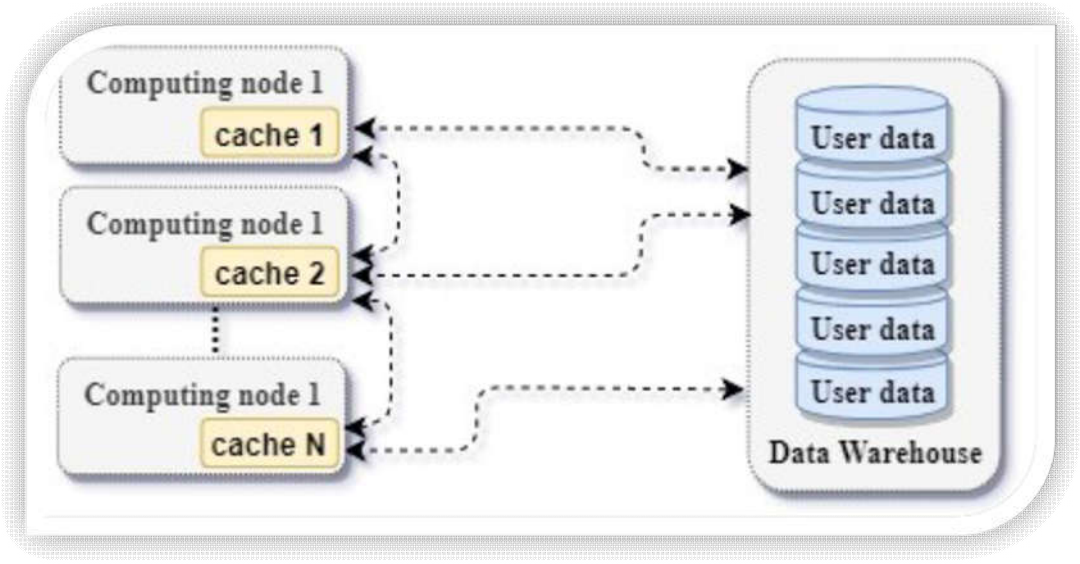
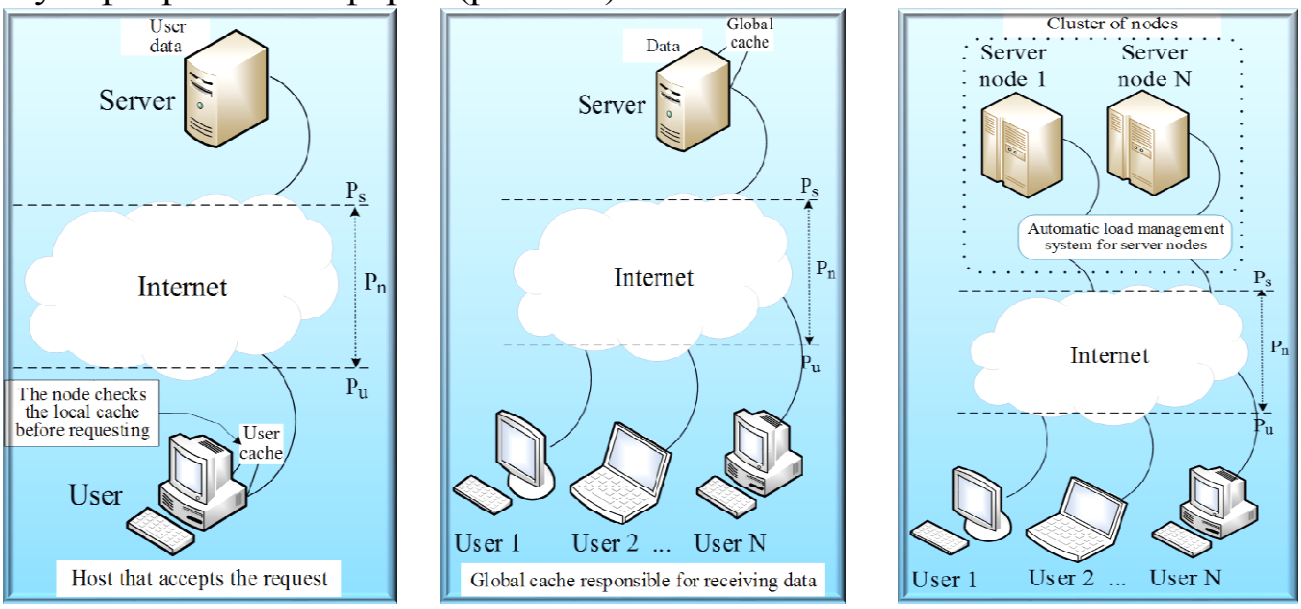


Рисунок 4.5 – Схема використання розподіленого кешу

Узагальнені схеми використання розробленої програмної системи з використанням хмарної технології на незалежних обчислювальних вузлах із розподілом навантажень (балансувань). Балансування навантаження (Load Balancing) програмної системи застосовується для оптимізації виконання обчислень за допомогою розподіленої високонавантажувальної системи, із боку користувача, так і на обчислювальному сервері чи платформі (рис. 4.6).



P_n – переміщення навантаження на сервер або до користувача (тонкий або товстий клієнт); P_s – навантаження на сервер або кластер; P_u – навантаження на комп'ютер користувача.

Рисунок 4.6 – Схеми розподілених обчислень та використання кешів для балансування навантажень

Балансування навантаження на обчислювальні вузли передбачає рівномірне навантаження апаратно-програмних систем на незалежних обчислювальних платформах. Розроблена програмна система, що балансує навантаження, повинна вирішувати, на якому обчислювальному вузлі потрібно виконувати обчислення, які пов'язані з новим інформаційним завданням. Отже, головною задачею балансування – забезпечення процесу перенесення (migration – міграція) частини або всього обчислення із найбільш завантажених обчислювальних вузлів на менш завантажені.

4.4. Математичні моделі та методи балансування навантаження у розподілених програмних системах

На незалежних обчислювальних платформах, де розміщений Web-сервер, який володіє обмеженим ресурсом потужності, тобто може виконати лише певне число операцій за одиницю часу (секунду). Отож, архітекторам програмних систем з використанням хмарної технології потрібно пам'ятати про те, що для отримання відповіді на запит необхідно виконати декілька обчислювальних операцій. Тому Web-сервер може опрацьовувати за одиницю часу лише кінцеве число користувацьких запитів, кількість яких визначає обчислювальна потужність Web-сервера [327]. Якщо за визначену одиницю часу кількість користувацьких запитів, які надійшли у систему для опрацювання, перевищує обчислювальні потужності сервера, то деякі залишаться не виконаними або втраченими. Для того щоб зменшити їхню кількість необхідно збільшити обчислювальну потужність Web-сервера. Як правило для вирішення питання недостатності обчислювальної потужності Web-сервера використовується лінійне збільшення потужності обчислювального вузла шляхом простого додавання ще одного Web-сервера. Такий архітектурний підхід до вирішення питання недостатності потужності обчислювального вузла має свої недоліки: по-перше, потужність наявних обчислювальних машин апаратно обмежена, а по-друге, їхня вартість обчислювальних машин зростає значно швидше, ніж їх продуктивність, тобто загальна вартість обчислювальної машини, яка має у два рази більшу апаратну потужність зросте значно більше ніж у два рази.

Проте існує й інший підхід до вирішення вище наведених проблем, який полягає у тому, щоб використати для обслуговування (опрацювання) користувацьких запитів кілька Web-серверів. За такого архітектурного підходу у розробці програмних систем з використанням хмарної технології вартість управлінського рішення буде прямо пропорційна зростанню його загальної потужності, тоді як верхня межа обчислювальної потужності визначається не рівнем застосованої апаратної технології, а загальною кількістю залучених до обчислення Web-серверів.

Зазвичай безліч комп'ютерів, які об’єднані між собою за певним технологічним принципом та спільно опрацьовують користувацькі запити, називається Web-фермою, а комп'ютери, що належать до Web-ферми, називаються реальними серверами опрацювання інформаційних потоків даних. Згідно із парадигмою організації глобальної інтернет-мережі, інформаційний запит користувача, який направляється на певну адресу обчислювального вузла, може отримати лише один комп'ютер. Отже, для того, щоб інформаційні користувацькі запити, що направляються на адресу Web-сервера, могли опрацьовувати декілька реальних обчислювальних серверів на вузлі, який приймає ці запити, необхідно запустити спеціальний сервіс, який називається хмарним сервісом балансування навантаження. Хмарний сервіс балансування навантаження призначений для виконання таких основних задач:

- приймається та опрацювання запитів від користувача;
- вибір реального Web-сервера, який буде опрацьовувати цей інформаційний запит від користувача;
- перенаправлення інформаційного запиту від користувача до реального Web-сервера;
- приймання відповіді реального Web-сервера на користувацький інформаційний запит;
- перенаправлення відповіді реального Web–сервера на інформаційний запит користувачу.

Отже, всі інформаційні запити користувачів, які потрапляють до Web-ферми, спочатку опрацьовує хмарний Web-сервіс балансування апаратно-програмного обчислювального навантаження. Для опрацювання одного інформаційного запиту користувача хмарному Web-сервісу балансування навантаження потрібний певний процесорний час [112, 113]. Тому загальна потужність Web-сервера обмежена не тільки обчислювальною потужністю ферми, а й потужністю певного обчислювального вузла, на якому запущено хмарний сервіс балансу-

вання навантаження. Відповідно до досліджень можна з’ясувати, наскільки важливо реалізувати при розробці програмної системи з використанням хмарної технології на незалежних обчислювальних платформах максимально ефективний сервіс для балансування обчислювального навантаження.

За типом використання програмно-апаратних засобів можна виділити два типи Web-ферм, таких як симетричні та асиметричні. Симетричною можна назвати Web-ферму, якщо всі її сервери спроможні опрацьовувати однотипні інформаційні запити користувачів. Асиметричною можна назвати Web-ферму, якщо кожен із її серверів створений для опрацювання певного класу інформаційних запитів користувачів. Наприклад, якщо користувач застосовує HTTP-сервіс, один сервер з асиметричною архітектурою Web-ферми, може надати інформаційні запити користувачів на текстові html-сторінки, а інший Web-сервер спеціалізуватися на опрацюванні інформаційних запитів користувачів на мультимедійні або графічні файли.

Для симетричних Web-ферм хмарний сервіс балансування обчислювального навантаження не повинен визначати, який саме ресурс потребує користувач, оскільки всі реальні Web-сервери мають однаковий набір програмно-апаратних ресурсів. Тому хмарний сервіс балансування обчислювального навантаження повинен прийняти відповідне з'єднання і, не чекаючи надходження інформаційного запиту користувача, вибрати довільний реальний Web-сервер для обслуговування. Незважаючи на простоту архітектурного рішення організації симетричної Web-ферми, вона достатньо складна в обслуговуванні. Адміністратор хмарної платформи повинен постійно дбати про те, що б всі Web-сервери мали однакове інформаційне та програмне наповнення, такий підхід до вирішення хмарного балансування обчислювального навантаження є достатньо складним процесом з технічної точки зору [159-162]. Отже, запропоноване програмно-апаратне рішення потребує додаткових витрат довготривалої пам'яті на незалежних обчислювальних платформах, оскільки копія кожного інформаційного ресурсу має бути на кожному Web-сервері.

4.4.1. Математична модель організації обчислювального кластера розподіленої програмної системи

Формальну постановку задачі балансування хмарного навантаження можна провести, використовуючи моделі замкненої мережі масово-

го обслуговування (ММО) з центральним обслуговувальним вузлом. Ця модель дає змогу відобразити архітектуру розробленого кластера (рис. 4.7).

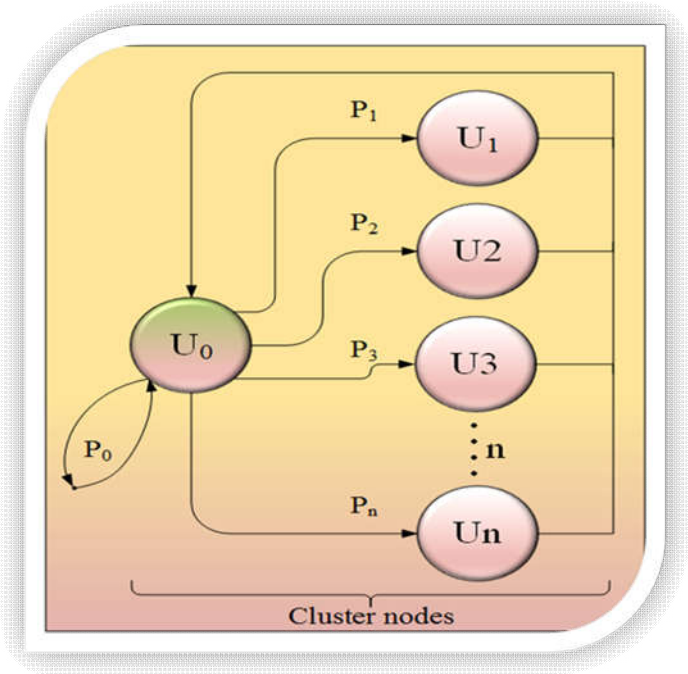


Рисунок 4.7 – Модель організації кластера з центральним обслуговуючим вузлом

Дана математична модель розробленої програмної системи з використанням хмарної технології на незалежних обчислювальних платформах дає змогу описати роботу мультипрограмної обчислювальної системи (з фіксованим числом обчислювальних вузлів), в якій допускається певна кількість інформаційних K завдань. Ці інформаційні завдання користувачів постійно циркулюють у розробленій програмній системі нескінченно довго, колективно використовуючи N її апаратно-програмних обчислювальних ресурсів. Центральний обслуговувальний вузол 0 – це кореневий обчислювальний вузол кластера, а решта $N_{1\text{-}n}$ вузлів – периферійні (зовнішні) вузли кластера. За такої архітектурної організації системи інформаційні завдання користувача циркулюють між обчислювальними вузлами кластера, водночас спочатку звертаючись до кореневого вузла, а вже потім до деякого периферійного вузла кластера з подальшим переходом до обслуговування у центральному вузлі і на останньому етапі знову звернувшись до певного периферійного вузла кластера тощо. Отже, упродовж життєвого циклу інформаційні завдання користувачів постійно повертаються до кореневого обчислювального вузла кластера. Перехідні ймовірності r_{ij} – це перехід

інформаційного завдання користувача у вузол *j* з вузла *i,* який представлено формулою (4.1):

$$r_{ij} = \begin{cases} p_j, & i = 1, \quad 1 \le j \le n \\ 1, & 2 \le i \le n, \quad j = 1, \\ 0, & \text{в інших випадках} \end{cases} \tag{4.1}$$

де r_{ij} – це ймовірності переходу завдання; *i* – номер вузла з якого переходить завдання; *j* – номер вузла, до якого переходить завдання; *p* – завдання.

$$\sum_{j=1}^{N} p_j = 1. \tag{4.2}$$

У реальній обчислювальній ситуації більшість завдань зрештою залишають розроблену програмну систему з використанням хмарної технології, і в моменти їхнього виходу із системи, у систему по одному входять все нові інформаційні завдання користувачів. Такий алгоритм враховано у моделі шляхом дозволу інформаційним завданням прямого повернення в кореневий обчислювальний вузол (з ймовірністю p_1), що на практиці означає вивільнення старого інформаційного завдання користувача і надходження замість нього оновленого. У такому випадку нова вимога на обслуговування в кореневому обчислювальному вузлі є заявкою на заміну інформаційного завдання користувача. Отже, число інформаційних завдань користувачів у системі залишається постійним і дорівнює кількості *K*.

У запропонованій моделі з центральним обслуговуючим обчислювальним вузом, де у кожному вузлі перебуває один обслуговувальний вузол ($m_i=1$), а час обслуговування в *i*-му вузлі розподілено за показовим (експоненційним) законом з параметром μ_i

Нехай матриця *R* – матриця перехідних ймовірностей $R(r_{ij})$ між обчислювальними вузлами мережі, тоді випадок з центральним обслуговуючим обчислювальним вузлом можна описати у такий спосіб:

$$R = \begin{vmatrix} p_1 & p_2 & p_3 & \dots & p_n \\ 1 & 0 & 0 & \dots & 0 \\ 1 & 0 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots \\ 1 & 0 & 0 & \dots & 0 \end{vmatrix}, \tag{4.3}$$

де *R* – матриця перехідних ймовірностей $R(r_{ij})$; p_1 – ймовірність переходу.

Виконання завдання обмежується розв'язанням рівняння $\lambda=\lambda R$. Для розглянутої матриці розв'язання можливе у такий спосіб:

$$\mu_i x_i = \begin{cases} \mu_1, & i = 1 \\ \mu_1 p_i, & i = 2,3,\dots,n \end{cases}, \tag{4.4}$$

де μ_i – параметр показового (експоненційного) закону; i – номер обчислювального вузла, з якого переходить завдання; p_i – ймовірність переходу.

Загальне математичне рішення для будь-якої замкнутої Марковської мережі масового обслуговування набуде такого вигляду:

$$p(k_i, k_2, \dots, k_n) = \frac{1}{G\left(K \sum_{k \in A} \prod_{i=1}^{n} \left(\frac{\mu_1 p_i}{\mu_i}\right)^{k_i}\right)} \prod_{i=1}^{n} x_{x_i}^{k_i}, \tag{4.5}$$

де p – загальна ймовірність переходу; k_i – кількість обчислень для перенесення; K – кількість обчислювальних завдань у системі; μ_i – параметр показового закону; p_i – ймовірність переходу в i-й вузол.

Для окремого випадку рішення набуває такого вигляду:

$$p(k_i, k_2, \dots, k_n) = \frac{1}{\sum_{k \in A} \prod_{i=1}^{n} \left(\frac{\mu_1 p_i}{\mu_i}\right)^{k_i}} \prod_{i=1}^{n} \left(\frac{\mu_1 p_i}{\mu_i}\right)^{k_i}, \tag{4.6}$$

де p – загальна ймовірність переходу; k_i – кількість обчислень для перенесення; K – кількість завдань у системі; μ_i – параметр показового закону; p_i – ймовірність переходу в i-й вузол.

Проаналізувавши математичну формулу (4.6), отримаємо очевидне розв'язання завдання, яке виражається параметрами системи μ_i та p_i. Нехай тепер A_i – стаціонарна ймовірність того, що i-й вузол не порожній. У такому випадку можна показати, що:

$$A_i = \begin{cases} \dfrac{G(K-1)}{G(K)}, & i = 1 \\ \dfrac{\mu_1 p_1}{\mu_i} A_1, & i = 2,3,\dots,n \end{cases}, \tag{4.7}$$

де A_i - стаціонарна ймовірність того, що i-й вузол не порожній; K – кількість завдань у системі; μ_i – параметр показового (експоненційного) закону; p_i – ймовірність переходу в i-й вузол; i – номер вузла з якого переходить завдання для опрацювання.

Відповідно до аналізу математичної формули можна стверджувати, що інтенсивність, з якою завдання надходять у i-й обчислювальний вузол, дорівнює інтенсивності, з якою вони його залишають. Ця рів-

ність показує, що міра, яка визначає, наскільки вузьке місце створюється в *i*-му обчислювальному вузлі, пропорційна швидкості зміни продуктивності залежно від зростання інтенсивності обслуговування в цьому обслуговувальному вузлі. Водночас продуктивність програмної системи з використанням хмарної технології на незалежних обчислювальних платформах визначається, як середня кількість обчислювальних завдань, які потрібно опрацювати.

Ґрунтуючись на результатах наведеного математичного аналізу, розглянемо обчислювальну задачу балансування навантаження у серверній Web-фермі. Типовий користувацький обчислювальний запит складається з назви ресурсу (URL, імені файла, номера інформаційного повідомлення). Водночас до того, як користувач відправить через програмну систему з використанням хмарної технології ім'я ресурсу Web-сервера, між ними може відбутися обмін потоками даних додатковими повідомленнями та правильний погляд інтернет-протоколів (наприклад, з метою авторизації або ідентифікації користувача). Уся сукупність інформаційних повідомлень (разом із запитами та відповідями на них) при встановлені з'єднання між користувачем та Web-сервером, які закінчуються розривом цих з’єднань, називається завершенням програмної сесії користувача в обчислювальній системі.

У разі використання архітектури асиметричної Web-ферми, кожен реальний обчислювальний сервер не зобов'язаний вміти відповідати на будь-який користувацький запит. Але для кожного користувача обчислювального запиту у Web-фермі повинен бути реальний Web-сервер, здатний опрацьовувати ці інформаційні запити (наприклад, всі URL типу gif можуть зберігатися на одному реальному Web-сервері, а URL типу html – на іншому). Отже, хмарний сервіс обчислювального балансування навантаження повинен на підставі користувацького запиту підібрати реальний Web-сервер, який може його опрацьовувати цей запит і перенаправляти йому запити.

Для обміну користувацькими повідомленнями високорівневого протоколу використовується протокол менш високого рівня. У випадку використання мережі Інтернет – це протокол управління передачею. Якщо інформаційний потік даних відправлений за його допомогою протоколу управління передачею, то гарантується його безпомилкова доставка. Користувацькі повідомлення високорівневих протоколів обміну інформаційних потоків даних, де користувацькі запити та їх відповіді Web-сервера є у термінах протоколу управління передачею, трактуються як такі дані, які необхідно доставити.

Щоб передати інформаційні потоки даних від одного обчислювального комп'ютера іншому, користувач користуючись протоколом обміну управління передачею даних, то потрібно встановити відповідне з'єднання між цими комп'ютерами. При відправлені інформаційних потоків даних за допомогою протоколу управління передачею вони розбиваються на пакети, і кожен пакет надсилається індивідуально. На іншому кінці з'єднання пакети збираються у відповідному порядку, і користувач отримує ту ж послідовність інформаційного потоку даних, яку посилав йому відправник. До кожного індивідуального пакета інформаційних даних формується додатковий заголовок – заголовок протоколу обміну управління передаванням даних, у якому вказується консолідована інформація, що дає змогу упорядковувати різноманітні пакети правильно [196]. Так само протоколу обміну управління переданням даних входить заголовок з інформацією, що сприяє виявленню спотворення даних – це так звана контрольна сума пакета, взята із цього заголовка та його загального вмісту. Якщо він спотворюється, то контрольна сума, підрахована одержувачем, не збігається із контрольною сумою, вказаною в заголовку пакета. Також до заголовка пакета входять спеціальні поля, які називаються опціями, а їхній набір визначається при встановленні з'єднання між обчислювальними вузлами. Набір цих опцій переважно слугує для оптимізації швидкості передачі інформаційних потоків даних, а їхній набір може залежати від способу реалізації протоколу обміну управління переданням даних. Цей протокол сам послуговується протоколом нижчого рівня – інтернет-протоколу.

Отже, такий спосіб реалізації взаємодії між обчислювальними вузлами, коли один протокол користується послугами іншого протоколу, називається стеком. Зазвичай протокол-стек реалізується, як частина ядра операційної системи. Сервіси, що створюють протоколи більш високого рівня (HTTP, FTP, POP3), зазвичай утворюються в адресному просторі користувача. Водночас хмарні сервіси, які реалізують високорівневі протоколи, застосовують функції протоколу обміну управління переданням інформаційних потоків даних за допомогою спеціальних системних викликів. Тобто встановлення з'єднання між обчислювальними вузлами, читання та надсилання інформаційних потоків даних через протокол обміну управління переданням відбуваються шляхом системних викликів.

При роботі в визначеному для користувача адресному просторі хмарний сервіс, який для своєї діяльності реалізує один із протоколів,

отримує доступ лише до інформації збереженої у цьому адресному просторі обчислювального вузла. Усі дії щодо розподілення інформаційних потоків даних на транспортні пакети з підрахунком контрольних сум та формуванням відповідних заголовків пакетів відбуваються всередині протоколу обміну управління передачею інформаційних потоків даних і з призначеного для користувача адресного простору обчислювального вузла, тому до цієї інформації користувач не має оперативного доступу.

4.4.2. Метод балансування обчислювального навантаження програмних систем

Суть запропонованого методу для балансування навантаження на обчислювальних вузлах полягає у тому, що користувацький інформаційний запит та відповідь на нього формує клієнтська прикладна програма і Web-сервер за допомогою одного з відомих високорівневих протоколів (HTTP, FTP, POP3). На практиці це означає, що користувацький інформаційний запит та відповідь на нього є сформованою синтаксично правильною послідовністю байтів із погляду правил, встановлених обміну потоків даних.

Ґрунтуючись на аналіз досліджень можна зробити висновок про те, що хмарний сервіс балансування обчислювального навантаження повинен бути реалізований як частина ядра операційної системи. Такий науково-практичний підхід забезпечує значну перевагу в обчислювальній продуктивності розробленої програмної системи з використанням хмарної технології завдяки таким причинам:

1. Для прийому або відправлення потоків даних не потрібно проводити перемикання між адресним простором користувача та адресним простором ядра операційної системи. Ця операція з погляду потребує чималих, а застосовуючи хмарний сервіс повністю у просторі ядра операційної системи, ми можемо мінімізувати або повністю її уникнути.

2. Хмарний сервіс регулювання обчислювального навантаження, який реалізований в адресному просторі користувача, не має доступу до інформаційного потоку даних, які зберігаються в адресному просторі ядра операційної системи. Отже, будь-яке відправлення чи отримання інформаційних потоків даних потребує виконання додаткового копіювання цих даних з адресного простору користувача в адресний простір ядра операційної системи (рис. 4.8). Завдяки хмарному сервісу, як

частині ядра операційної системи розроблена програма повністю уникає дублюванню інформаційного потоку даних між адресними просторами. Запропонований підхід значно знизить обчислювальне навантаження на процесор та шину пам'яті розробленої програмної системи.

3. Чимала частина інформаційного потоку даних (наприклад, відповідь користувачу) проходить через хмарний сервіс для балансування обчислювального навантаження транзитом, тобто потоки даних не опрацьовує сервіс, а вони тільки пересилаються з одного обчислювального вузла в інший. Маючи доступ до заголовків протоколу обміну управління передачею інформаційних пакетів, можна оптимізувати механізм переадресації потоку даних з одного обчислювального вузла в інший, використавши їх інформацію з протоколу про записане значення контрольної суми інформаційного пакета. Доступ до неї можливий тільки з адресного простору ядра операційної системи.

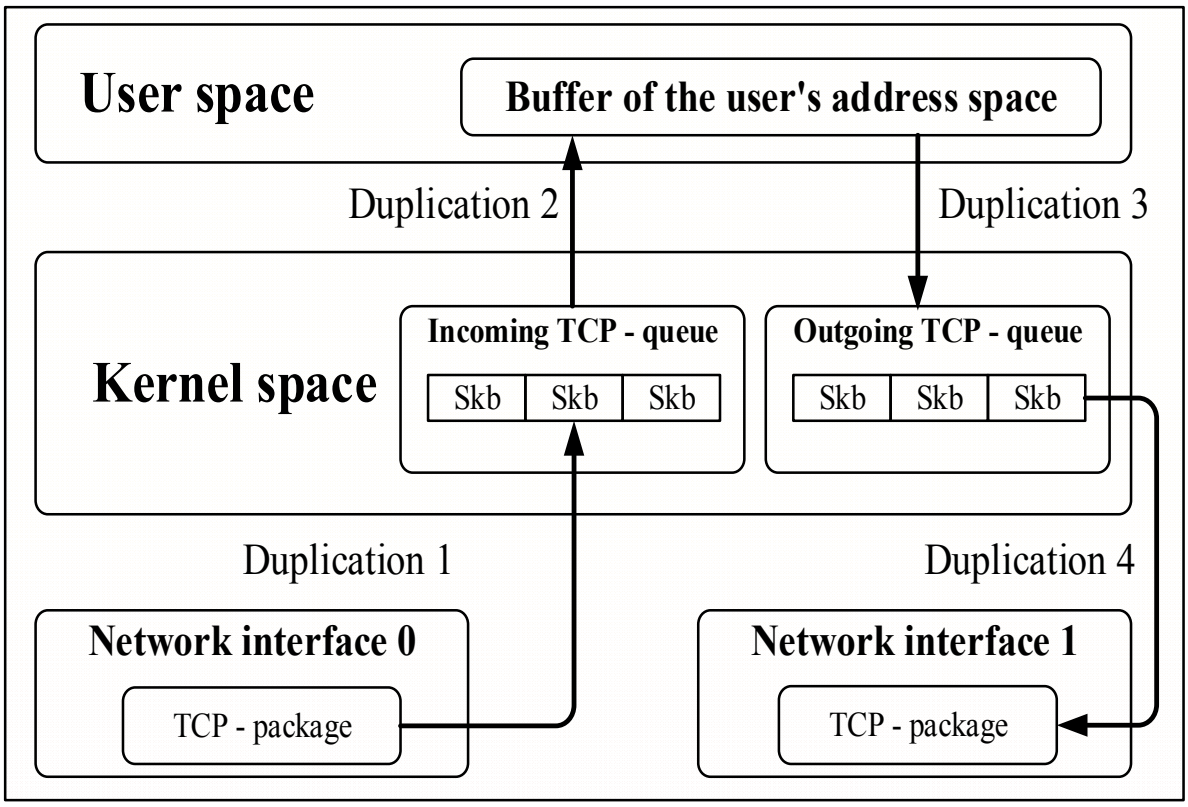


Рисунок 4.8 – Балансування запитів, що виконує хмарний сервіс, який розташований в адресному просторі користувача

Для отримання та відправлення інформаційного потоку даних використовуються функції протоколу обміну управління переданням інформаційних потоків – стеків розробленої програмної системи, які працюють із ними на рівні інформаційних пакетів. Це означає, що хма-

рний сервіс забезпечує балансування обчислювального навантаження не повинен безпосередньо звертатися до інших мереживних пристроїв на незалежних обчислювальних платформах для отримання або відправлення цього пакета, а лише використовувати для цієї операції протокол обміну управління переданням інформаційних потоків-стеку. Причому функції повинні оперувати інформаційними потоками даних як послідовністю пакетів, а не як безперервним інформаційним потоком. Цю обчислювальну операцію опрацювання інформаційних потоків даних потрібно робити з огляду на такі причини (рис. 4.9):

1. Інформація що міститься в отриманих інформаційних пакетах з протоколу обміну управління передачею інформаційних потоків даних необхідно оптимізувати їх відправлення в інше з'єднання та на інший обчислювальний вузол. Якщо користуватися функціями, що працюють на рівні обміну даних із буферів, то наявна інформація про контрольні суми у заголовках інформаційних пакетів буде недоступна.

2. Отримуючи дані як послідовність інформаційних пакетів з протоколу обміну управління переданням інформаційних потоків, хмарний сервіс може використовувати ці пакети при відправці даних в інший обчислювальний вузол. Завдяки цьому підходу ми уникаємо протиріч щодо створення додаткового інформаційного буфера обміну потоками даних для пакета та копіювання його вмісту.

3. Використання протоколу обміну управління переданням інформаційних потоків даних, який переносить їх у стек для подальшого опрацювання за правилами виставленим опцій у заголовках цих пакетів. Якщо відправляти інформаційні пакети безпосередньо у мережевий обчислювальний пристрій, послідовність опцій буде задана не правильно водночас, а це у значно знизить швидкість опрацювання та переданням інформаційних потоків даних до нових обчислювальних вузлів.

Інформаційні пакети протоколу обміну управління переданням інформаційних потоків, що надійшли від користувача, повинні зберігатися у внутрішній черзі операційної системи доти, поки не приймуть користувацький запит та не буде команда використовувати ці їх відправлення запиту обраному на його основі реальному Web-сервері.

Така організація управління пакетами з інформаційними потоками даних дає змогу користувачу виконати такі дії:

- проаналізувати інформаційний запит користувача, навіть якщо він був розбитий на кілька послідовних пакетів обміну інформаційними потоками даних;

- використовувати пакети з інформаційними потоками даних при відправленні запиту на незалежний обчислювальний вузол, що дає змогу уникнути додаткового створення та копіювання пакетів, а також забезпечити оптимізацію обчислення контрольної суми цього пакета.

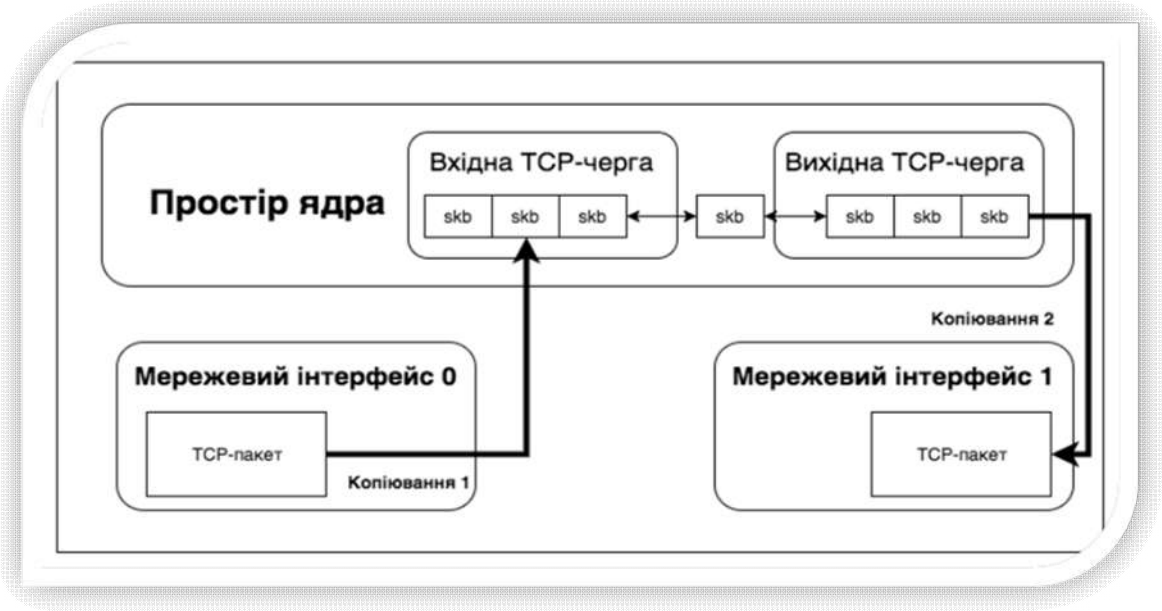


Рисунок 4.9 – Балансування запитів у сервісі розташованому в адресному просторі ядра операційної системи

Для вибору обчислювального вузла, який буде опрацьовувати користувацький інформаційний запит з потоку даних, необхідно користуватись заздалегідь визначеними правилами. Вони можуть містити таку інформацію:

1. Вміст користувацького інформаційного запиту для опрацювання потоків даних, які застосовують інформацію, за допомогою якої можна вибирати незалежний обчислювальний Web-сервер залежно від потрібного ресурсу. Використання цих правил дозволяє використовувати в опрацюванні даних асиметричну модель Web-ферми.

2. IP-адресу користувача, що здійснює інформаційний запит для опрацювання потоку даних, використовуючи доступну інформацію, яку можна виділити у привілейованих групах користувачів для забезпечення кращої якості обчислювального обслуговування, що забезпечить розподілення запитів на більш потужніший і менш завантажений Web-сервер.

3. Величину завантаження обчислювальних процесорів та кількість використаної пам'яті Web-серверів. Використовуючи відомості про інформаційні потоки даних для опрацювання користувацького за-

питу, хмарний сервіс балансування обчислювального навантаження може автоматично вибрати найменш завантажений Web-сервер, водночас значно збільшити швидкість опрацювання запиту.

Для формування та надсилання пакетів протоколу обміну управління передачею інформаційних потоків даних необхідно використовувати алгоритм автоматизованого перерахунку його контрольної суми цього пакету, який дозволяє контролювати коректність відправленого та отриманого пакета обміну даними через контрольну суму цих даних яка записана у заголовку цього пакета. У пакеті протоколу обміну управління передачею інформаційних потоків, який отримує хмарний обчислювальний сервіс для опрацювання балансування навантаження з боку Web-сервера або користувача, контрольну суму автоматично порахував відправник.

Ідея запропонованого алгоритму перерахунку контрольної суми пакета обміну інформаційними даними полягає у тому, щоб використовувати при автоматичного розрахунку контрольної суми вихідного пакета відповідну контрольну суму вхідного пакета обміну даними. Отже, це можливо виконати лише завдяки тому, що контрольна сума у певною мірою «лінійно» залежить від буфера обміну даними, для якого вона обчислюється. Якщо буфер обміну даними розбити на дві незалежні частини, то його контрольна сума лінійно виразиться через контрольні суми частин цього пакета. У вихідному інформаційному пакеті обміну даними змінюється тільки заголовок, а для того щоб отримати його контрольну суму достатньо обчислити лише контрольну суму його заголовка. Якщо типова довжина протоколу обміну управління переданням інформаційних потоків заголовка становить 60 байтів, а довжина області потоку даних – 1500 байтів, то використання обчислювальних ресурсів при застосуванні алгоритму перерахунку знижується у двадцять п’ять разів.

4.4.3. Оцінка ефективності розроблення розподіленої програмної Web-системи

При розробленні та аналізі високонавантаженої розподіленої відмовостійкої програмної Web-системи основоположним моментом є ефективність їхнього використання, яка визначається як частка від ділення суми усіх вихідних параметрів на суму всіх вхідних факторів, які впливають на неї. Для кожного незалежного інформаційного обчислювального модуля чи вузла обчислюється величина ефективнос-

ті, а отже, проводиться порівняння спостережень за продуктивністю програмної Web-системи. Їх можна виконати за допомогою методу звичайного лінійного програмування при використанні різних концептуальних моделей та їхніх різновидів [16]. Для цього користувач визначає загальну кількість залучених обчислювальних модулів (вузлів) на незалежних хмарних платформах шляхом побудови заданого критерію ефективності, а для усіх інших обчислювальних вузлів – показника їхнього неефективного використання. Але водночас базисним пунктом є те, що критерії ефективності або неефективності розробленої програмної Web-системи визначаються на основі цього рішення.

Отже, критерій ефективності дорівнює співвідношенню суми загальних зважених результатів продуктивності Web-системи до суми використаних інформаційних ресурсів розробленої програмної системи на основі хмарної технології за умовою (4.8):

$$E = \frac{\sum y}{\sum x}, \tag{4.8}$$

де E – розрахунковий критерій ефективності розробленої програмної системи; y – зважений вихідний параметр; x – зважений вхідний параметр.

Оцінка відповідних величин критеріїв ефективності відбувається за допомогою оптимізації роботи розробленої програмної системи з використанням хмарної технології. Критерієм для виявлення ефективності є досягнення оптимуму Парето, або, відповідно, ефективністю Парето. Цей ефект широковідомий та застосовується в економічній теорії, проте його застосовують також для оцінки і побудови високонавантажених розподілених програмних систем. Відповідно до визначення оптимуму Парето, Web-система цілком є на 100% ефективна, якщо:

- жоден із вихідних параметрів не може бути підвищений без підвищення одного або більше вхідних факторів та зниження інших вихідних параметрів;
- жоден із вхідних факторів не може бути зменшений без зниження одного або збільшення вихідних параметрів та підвищення інших вхідних чинників.

Отже, дані які стосуються визначення лише понять відносних критеріїв ефективності не можуть бути таким жорсткими у своїй визначеності, оскільки справжній критерій ефективності розробленої програмної системи у більшості випадків невідомий. Для забезпечення 100%

відносної ефективності розробленої програмної Web-системи необхідно досягнути позитивного результату порівняно з іншими відповідними Web-системами, для яких не існує відповідне твердження про неефективність по відношенню, до одного або декількох вхідних та вихідних факторів. Порівнюючи ефективність Web-систем, розглядають Web-системи, які при однотипних вхідних або вихідних факторах переслідують однакові цілі, для опрацювання інформаційних потоків даних розробленої програмної системи з використанням хмарної технології. Відповідно до вище викладених міркувань, в основі методики визначення критерію ефективності Web-систем може лежати ідея відносної ефективності.

Вимірювання критерію ефективності Web-систем відбувається на основі оптимального зваженого співвідношення між використаними вихідними параметрами та вхідними факторами. Запропонований метод дає змогу визначати інтегральну оцінку параметрів критеріїв ефективності у такий спосіб, щоб обчислювальні модулі (вузли) на незалежних платформах, де розміщена Web-система, яку необхідно оцінити, перебували у ймовірнісному діапазоні від 0 (мінімальний критерій ефективності) до 1 (максимальний критерій ефективності). З огляду на співвідношення вхідних і вихідних параметрів всіх спостережуваних обчислювальних одиниць дослідження потрібно вибирати, якщо можливо вищі значення критерію ефективності e_0. Водночас інтегральна оцінка усіх залучених досліджень обчислювальних модулів (вузлів) проводиться у такий спосіб, щоб вона мала максимальну величину критерію ефективності, не перевищуючи граничного значення – 1. Формально ці дії полягають у розв'язанні завдання максимізації, умова (4.9):

$$e_0 = \frac{\sum_{j=1}^{s} v_j x_j}{\sum_{i=1}^{r} u_i y_i} \to max!, \qquad (4.9)$$

де $\frac{\sum_{i=1}^{s} v_j x_{jm}}{\sum_{j=1}^{r} u_i y_{jm}} \leq 1$ для всіх модулів $m = 1, 2, ...n$; e_0 – значення критерію ефективності досліджуваної Web-системи; n – число одиниць, які порівнюються; r – число вхідних факторів; s – число вихідних параметрів; x_i – вираз i-й вхідний фактор досліджуваного обчислювального модуля (вузла); y_j – вираз j-й вихідний параметр досліджуваного обчислювального модуля (вузла); x_{im} – вираз i-того вхідного фактору m – того обчислювального вузла з $i = 1, ..., r$ і $m = 1, ..., n$; y_{jm} – вираз j-й вихідний параметр обчислювального вузла m з $i = 1, ..., r$ і $m = 1, ..., n$; v_i – зважу-

вання вхідного фактору i з $i = 1, ..., r$; u_j – зважування вихідного параметра j з $j = 1, ..., s$.

При вирішенні завдання максимізації критерію ефективності виникає нагальна проблема наявності частки у розподілі двох лінійних агрегованих значень. Також проблеми максимізації критерію ефективності Web-систем називаються лінійним програмуванням частки. Водночас є безліч можливостей трансформації лінійного програмування частки у проблему лінійного програмування.

$$f_0 = \frac{\sum_{i=1}^{r} v_i x_i}{\sum_{j=1}^{s} u_j y_j} \rightarrow min!, \tag{4.10}$$

де $\frac{\sum_{i=1}^{r} v_i x_i}{\sum_{j=1}^{s} u_j y_j} \geq 1$– для всіх модулів $m = 1,2,...n$; $u_j \geq 0, j = 1, 2, ..., s$; $v_i \geq 0, i = 1, 2, ..., r$; y_j – вираз j – й вихідний параметр досліджуваного обчислювального модуля (вузла); x_{im} – вираз i – того вхідного фактору m – того обчислювального вузла з $i = 1, ..., r$ і $m = 1, ..., n$; y_{jm} – вираз j – того вихідного параметра m – того обчислювального вузла з $i = 1, ..., r$ і $m = 1, ..., n$; v_i – зважування вхідного фактору i з $i = 1, ..., r$; u_j – зважування вихідного параметра j з $j = 1, ..., s$.

Для подальшого дослідження модифікуємо цю нелінійну проблему оптимізації критерію ефективності з ускладненим алгоритмом теорії дробового програмування (англ. Fractional Programing Theory) більш традиційним та менш ресурсно затратним алгоритмом лінійного програмування. Отже, ускладнена проблема оптимізації критерію ефективності може бути спрощена у лінійну проблему за допомогою методів лінійної оптимізації. Щоб отримати величину критерію ефективності для всіх обчислювальних модулів (вузлів), необхідно вирішити завдання максимізації індивідуально для кожного окремого модуля, який задіяний у науковому дослідженні. У цьому випадку вектори x_{im} і y_{im} замінюються щоразу за допомогою профілю вхідних та вихідних параметрів досліджуваного обчислювального модуля, відповідно. В іншому завданні максимізації критерію ефективності вони залишаються однаковими для кожного обчислювального вузла. У цьому заданні накладені обмеження, забезпечують знаходження якісних значень ефективності в області між нулем та одиницею ($e_0 \in [0,1]$).

З критерію обмежень ефективності формується основна властивість цієї моделі. Її цільова функція пропорційно намагається збільшити вихідний параметр спостережуваного обчислювального модуля до межі критерію ефективності. У математичному розв’язку визначеної

функції утворюється еквівалентна, або так звана охоплююча форма (англ. Envelopment form), що утворюється після застосування до неї теореми двоїстості. Принцип подвійності при використанні лінійного програмування свідчить про те, що для кожної розглянутої прямої лінійної моделі оптимізації критерію ефективності існує двоїста відповідна лінійна модель оптимізації, а у випадку рішення однієї моделі містить усі вирази для вирішення іншої моделі. Отже, вихідну оптимізовану модель будемо вважати прямою задачею лінійного програмування. Використовуючи двоїстий метод, ми мінімізуємо зважену суму вхідних параметрів щодо одного нормованого вихідного параметра. Математичне формулювання моделі лінійного програмування при використанні методу двоїстості здійснюється в такий спосіб:

$$\min g_0 = \sum_{i=1}^{r} t_i x_i, \tag{4.11}$$

де g_0 – величина критерію ефективності досліджуваного обчислювального вузла; x_i – вираз i – того вхідного фактору обчислювального вузла з $i = 1, ..., r$; t_i – змінні фактори зважування.

Лінійні комбінації визначають потенційні референтні групи для вимірювання критерію ефективності залучених до дослідження обчислювальних вузлів. Отже, відповідно до задачі оптимізації (4.11) цільової функції, яку мінімізують, то запропонований метод вибирає таку референтну групу, у якій критерій ефективності залученого обчислювального вузла виглядає недостатньо дієвий.

Отже, цю математичну проблему можна інтерпретувати так: щоб для дослідженого обчислювального вузла визначити мінімальний критерій ефективності вхідного параметра G_0, за якого порівняно зі зваженими ймовірностями порівняльних одиниць зважена комбінація вихідних параметрів за будь-якого вихідного параметра не зменшує вихідний параметр, а зважена інтегральна комбінація вхідних факторів за будь-якого вхідного фактору $\boldsymbol{i}$ в G_0 разів перевищує вхідний фактор.

4.4.4. Параметрична сіткова модель з часовими метриками для обчислення обсягу навантаження

Основні принципи роботи та функціонування сіткової моделі хмарних обчислень на незалежних обчислювальних платформах значно відрізняються від традиційних послідовних і паралельних моделей. Головною особливістю сіткової моделі хмарних обчислень є можливість

агрегування та колективного використання великих інформаційних наборів гетерогенних обчислювальних потоків даних, розподілених територіально. Здебільшого такий підхід надає значні переваги, наприклад коли розроблена програмна система вимагає інформаційних ресурсів, які недоступні у межах одного обчислювального вузла, то вона може їх отримати в інших обчислювальних вузлах, під'єднаних до хмарної мережевої сітки.

Проте використання такої складної архітектури опрацювання інформаційних потоків даних має декілька застережень та певних проблем. До високо гетерогенного, динамічно сформованого розподіленого обчислювального середовища досить складно застосовувати такі традиційні метрики визначення критерію продуктивності, як швидкість обчислень інформаційних потоків даних, пропускна здатність каналів обміну тощо. Тому для оцінки якості наданого хмарного сервісу необхідне використання спеціалізованих обчислювальних метрик. Припустимо, що у сітковому хмарному середовищі доступно m обчислювальних ресурсів і існує система розподілу потоку завдань t, яка забезпечує рівномірне розподілення інформаційних задач $j \in t$ на доступні ресурси. У межах використання такої розробленої програмної системи з використанням хмарної технології кожне інформаційне завдання користувача може бути розбите на певні обчислювальні дії $k \in j$. При постановці завдання визначається час опрацювання d_j необхідний, щоб отримати відповідний розв'язок. Кожне інформаційне завдання користувача j та всі відповідні дії $k \in j$ перебувають у хмарній сітці та в певний момент часу r_j. Отже, хмарна сітка, яка працює у режимі on-line, за користувацьких значень r_j, які є невідомі заздалегідь для значної кількості цих завдань. Як тільки надходить до хмарної обчислювальної мережі певна користувацька задача для опрацювання, то здійснюється планування її виконання та пошук і виокремлення необхідних ресурсів для запуску. Припустимо, що внаслідок фінального розподілення обчислювальних завдань S для кожної дії $k \in j$ необхідний певний час на опрацювання $C_k(S)$. Отже, обчислювальна задача користувача у хмарній мережі j може бути опрацьована не швидше, аніж за час, визначений виразом (4.12):

$$C_j(S) = \max_{k \in j} C_k(S), \tag{4.12}$$

де $C_k(S)$ – максимальний час опрацювання задачі користувача; k – дія; j – задача.

Визначимо p_j як час опрацювання завдання користувача $k \in j$. Отже, час опрацювання задачі користувача у хмарній мережі може бути обчислений у такий спосіб (4.13):

$$p_j = C_j(S) - \min_{k \in j}(C_k(S) - p_k), \tag{4.13}$$

де p_j – час опрацювання задачі користувача; $C_k(S)$ –час виконання; $C_j(S)$ – максимальний час виконання; p_k – час реалізації рішення.

Отримані інтегральні величини дають змогу оцінити властивості хмарного сіткового обчислювального середовища. Для проведення аналізу якості хмарного сервісу, що надається сітковим обчислювальним середовищем, то можна використати значення максимального запізнення користувацьких завдань (4.14):

$$L_{max} = \max_{j \in t}\left(C_j(S) - d_j\right), \tag{4.14}$$

де L_{max} – максимальне запізнення задачі користувача; $C_j(S)$ – максимальний час опрацювання; d_j – час невиконаних користувацьких задач.

Для проведення процесу оптимізації роботи розподіленого обчислювального хмарного середовища, необхідно досягти мінімізації показника значення L_{max}, а також можна використовувати значення показника T_j, який визначає, скільки завдань користувача спізнилося (4.15):

$$j \in t \wedge C_j > d_j, \tag{4.15}$$

де t – доступні інформаційні обчислювальні ресурси; d_j – час невиконаних користувацьких задач; C_j – час виконання однієї задачі; k – дія; j – задача.

Отже, цей показник надає інформацію про кількість невиконаних обчислювальних запитів користувачів, які надійшли у хмарну мережу. Споживання обчислювальних ресурсів хмарної мережі RC_k є певною підзадачею обчислень, які визначається як добуток відповідного часу розв'язання на кількість використаних ресурсів у мережі (4.16):

$$RC_k = p_k \times m_k, \tag{4.16}$$

де RC_k – загальне споживання обчислювальних ресурсів мережі; p_k – час реалізації рішення; m_k – кількість використаних ресурсів.

Використавши подане значення інтегрального споживання обчислювальних ресурсів хмарної мережі, можна обчислити величину використання доступних інформаційних ресурсів U (4.17):

$$U = \frac{RC(S)}{m \times (\max\limits_{j \in t} C_j(s) - \min\limits_{j \in t}(C_j(S) - p_j)}, \quad (4.17)$$

де U – величина використання доступних інформаційних ресурсів; m – кількість ресурсів; p_j – час опрацювання j-ї задачі; $RC(S)$ – час загального споживання; $C_j(S)$ – час виконання j-ї задачі.

За визначеною величиною, яка характеризує критерій оптимально використовуються обчислювальних ресурси хмарної розподіленої мережі. У процесі опрацювання інформаційних запитів у хмарній розподіленій мережі, часто виникають такі ситуації, коли під час опрацювання завдання користувача відбувається збій програмної системи.

Тоді завдання користувача щодо опрацювання інформаційного запиту повинне бути запущене декілька разів для його успішного виконання. Оскільки користувачі та адміністратори можуть ставити різноманітні (і навіть конфліктні) вимоги до хмарної сіткової Web-системи, то досить складно підібрати відповідну метрику, яка була б універсальна та задовольняла всіх. З погляду користувача розробленої програмної системи з використанням хмарної технології можна виділити такі метрики середнього часу відповіді на запит (Average Response Time) та середнього часу очікування запиту (Average Wait Time):

$$ART = \frac{1}{|t|}\sum_{j \in t}(C_j(S)), \quad (4.18)$$

де ART – середній час відповіді; $C_j(S)$ – час виконання j-ї задачі; p_j – час реалізації j-ї задачі; t – доступні ресурси.

$$AWT = \frac{1}{|t|}\sum_{j \in t}\left(C_j(S) - p_j\right), \quad (4.19)$$

де AWT – середній час очікування; $C_j(S)$ – час виконання j – ї задачі; t – доступні ресурси.

Значення параметра ART характеризує середній час відповіді на інформаційний запит користувача, а саме наскільки швидко відбувається опрацювання завдань користувачів. Окрім того, значення параметра AWT дуже важливе для розробників алгоритму дій щодо відносно невеликих інформаційних завдань з опрацювання запитів. Отже, оптимальний та простий метод вимірювання справедливої ефективності використання інформаційних і апаратних ресурсів – це розрахунок девіації середньозваженого часу очікування на опрацювання запиту:

$$AWTD = \frac{1}{|t|}\sqrt{\sum_{j \in t}(C_j(S) - p_j)^2 - (\sum_{j \in t}\frac{C_j(S) - p_j}{|t|})^2}, \qquad (4.20)$$

де *AWTD* – девіація середнього часу очікування опрацювання інформаційного запиту користувача; *t* – доступні ресурси; $C_j(S)$ – час виконання *j* – ї задачі; p_j – час реалізації *j* – ї задачі.

Для того щоб добитись оптимальних результатів використання хмарної сітки, необхідно мінімізувати параметр *AWTD*. У сучасних хмарних сіткових Web-системах, можливість завершення опрацювання заданого обсягу завдань більш важливе, аніж прискорення роботи розподіленої високонавантажуваль-ної Web-системи, яке отримане за допомогою такого опрацювання. Потрібно відзначити, що інформаційні завдання користувача, які виконуються у хмарних сіткових середовищах, можуть мати достатньо складну архітектуру, ніж інформаційні завдання користувача, які виконуються у паралельних системах із традиційною архітектурою. Наприклад, інформаційні потоки користувацьких завдань мають більш складну логічну структуру, аніж пакети відповідних завдань). Використання харної сітки вимагає зміни таких понять, як помилки розробленої програмної системи: Web-сервіс, який спроєктований на хмарній сітковій моделі, генерує надходження повідомлення про помилку, як тільки настає ситуація, коли неможливе успішне виконання інформаційного завдання користувача. Наприклад, збій розробленої програмної системи з використанням хмарної технології може відбутися тоді, якщо неможливо знайти відповідні ресурси для виконання інформаційних обчислень або через завершення.

Використовуючи поняття відмовостійкості у високонавантажувальних програмних системах з використанням хмарної технології, можна визначити, як основну можливість збільшувати термін відтермінування часу появи як програмної, так і апаратної помилки, поки не вичерпався хоч якийсь шанс того, що робота по опрацюванню користувацького запиту успішно завершиться. Метрику завершеного обсягу роботи по опрацюванню запитів користувача у програмній системі з використанням хмарної технології (Workload Completion), яка формується як відношення успішно завершених користувацьких завдань до загального обсягу усіх запитів, поставлених планувальником хмарного сіткового середовища:

$$WC = \frac{\sum j \in t \wedge (j\ complited)}{|t|}, \qquad (4.21)$$

де *WC* – показник завершеного обсягу роботи по опрацюванню запитів користувача; *t* – доступні ресурси; *j* – задача.

Ця метрика дає змогу визначити основні обмеження хмарної сіткової програмної системи, а її максимізація може бути головною ціллю. Проте, вона має і деякі критичні обмеження з позиції застосування вільних інформаційних та апаратних ресурсів, оскільки завдання користувача з меншою кількістю обчислювальних дій мають значно більший вплив на зміну цієї величини.

Метрика завершення дій з опрацювання завдань користувача (Task Completion) обчислює показник кількості завершених дій до загальної кількості виконуваних, які реалізувались у межах хмарної програмної системи розподілу користувацьких завдань:

$$TC = \frac{\sum j \in t \wedge k \in j \wedge (k\ complited)}{\sum j \in t\ |j|}, \tag{4.22}$$

де *TC* – показник завершених дій по опрацюванню завдань користувача; *t* – доступні ресурси; *k* – дія; *j* – задача.

Також варто розглянути таке поняття, як завершення розблокованих дій із завдань користувача на опрацювання (Enabled Task Completion), тобто коли вони можуть бути виконані лише тоді, як усі відповідні залежності для даної послідовності дій будуть виконані програмною системою з використанням хмарної технології:

$$ETC = \frac{\sum j \in t \wedge k \in j \wedge (k\ complited)}{\sum j \in t \wedge k \in j \wedge (k\ enabled)}, \tag{4.23}$$

де *ETC* – інтегральний показник завершених розблокованих дій щодо завдань користувача; *t* – доступні ресурси; *k* – дія; *j* – задача.

Отже, ми розглянули основні принципи функціонування хмарної сіткової моделі при розробці програмних систем, які суттєво відрізняються від традиційних послідовних і паралельних моделей. Основна їхня відмінність – це можливість агрегування і спільного використання великих наборів гетерогенних інформаційних ресурсів, розподілених між географічно-розділеними обчислювальними незалежними платформами. Здебільшого такі архітектурні підходи до розробки програмних систем з використанням хмарної технології приносять значні переваги та фінансові вигоди, наприклад, коли програмна Web-система вимагає значних інформаційних ресурсів, які недоступні в межах одного обчислювального вузла, вона може їх отримати в іншому вузлі, під'єднаному до хмарної сітки.

4.5. Моделі та алгоритми оптимізації архітектури розподіленої високонавантажувальної програмної Web-системи

Будь-які Webсистеми чи хмарні сервіси, які обслуговують чимало користувачів, апріорі високонавантажені. Однак високонавантажені розподілені програмні Web-системи не можуть ефективно застосовувати моделі, методи та алгоритми, які використовуються як основні підходи до розроблення звичайних Web-сайтів. Значне збільшення користувацької аудиторії та відповідних обчислень без належних підходів до оптимізації архітектури розроблення програмних систем з використанням хмарної технології з часом може призвести до значних ускладнень при їх обслуговуванні. Зараз усі розроблені та використані моделі, методи та алгоритми для розробки архітектури розподілених відмовостійких Web-систем застосовують загальноприйняті інформаційні технологій:

1. Глобальні та розподілені кеші. Ця інформаційна технологія та її варіації роблять потоки даних доступнішими, що на сьогодні найактуальніше для користувачів.

2. Статичне та динамічне балансування. Балансування обчислювального навантаження (Load Balancing) застосовується для оптимізації опрацювання розподілених (паралельних) обчислень. Ефективне балансування обчислювального навантаження на незалежних вузлах передбачає рівномірний розподіл завдань користувача на незалежних хмарних обчислювальних ресурсах. При появі нових користувацьких завдань розроблена програмна Web-система, що реалізує балансування, має прийняти управлінське рішення про те, якому обчислювальному вузлові необхідно виконувати опрацювання обчислень, які пов'язані з ними. Окрім того, ефективне балансування передбачає виконання інформаційного перенесення (migration – міграція) певної частини обчислень із завантажених обчислювальних вузлів на інші вузли, які менш завантажені.

3. Дублювання. Метод дублювання ґрунтується на використанні механізму контрольних точок. При використанні цього методу дублювання основний обчислювальний вузол періодично фіксує інформаційний стан Web-системи, тобто створює його теперішню копію. Водночас створюється і копія (snapshot) файлової системи. Після цього обчислювальний вузол продовжує працювати у звичному режимі, але з копією інформаційного стану, зробленою на підставі останньої копії.

4. Кластеризація. Кластеризація обчислювальної розподіленої програмної Web-системи полягає у тому, що одночасно декілька обчислювальних хмарних вузлів об'єднуються в одну програмну Web-систему, а за допомогою балансувального вузла вони рівномірно розподіляють обчислювальне навантаження між своїми ресурсами.

Особливість використання розробленого алгоритму у програмних Web-системах полягає у тому, що ці інформаційні технології використовують хмарні сіткові (grid) системи та технологію ієрархії відмов у своїй роботі. Представлення інформаційної обчислювальної моделі програмної Web-системи з балансувальним вузлом та загальним кешем, яка перенаправляє для опрацювання користувацький запит на найменш завантажений хмарний ресурс, використовуючи водночас глобальний кеш системи. Однак, потрібно підкреслити, що оптимізація використання високонавантажених Web-систем за допомогою хмарної сіткової моделі розподілення інформаційних потоків даних стає можливою лише тоді, коли кожен обчислювальний вузол Web-системи опрацьовує такий тип обчислень, як і інші вузли хмарної сітки.

Отже, як тільки балансувальний вузол отримав необхідний пакет із даними для опрацювання та відповіді користувацькі запити, то у глобальному кеші визначається найменш завантажений обчислювальний вузол. Основа цього процесу – непрямі інформаційні потоки даних, щоб суттєво обмежити обсяг комунікації між балансувальним та обчислювальними вузлами, які забезпечують обчислення згідно із запитом користувачів (рис. 4.10).

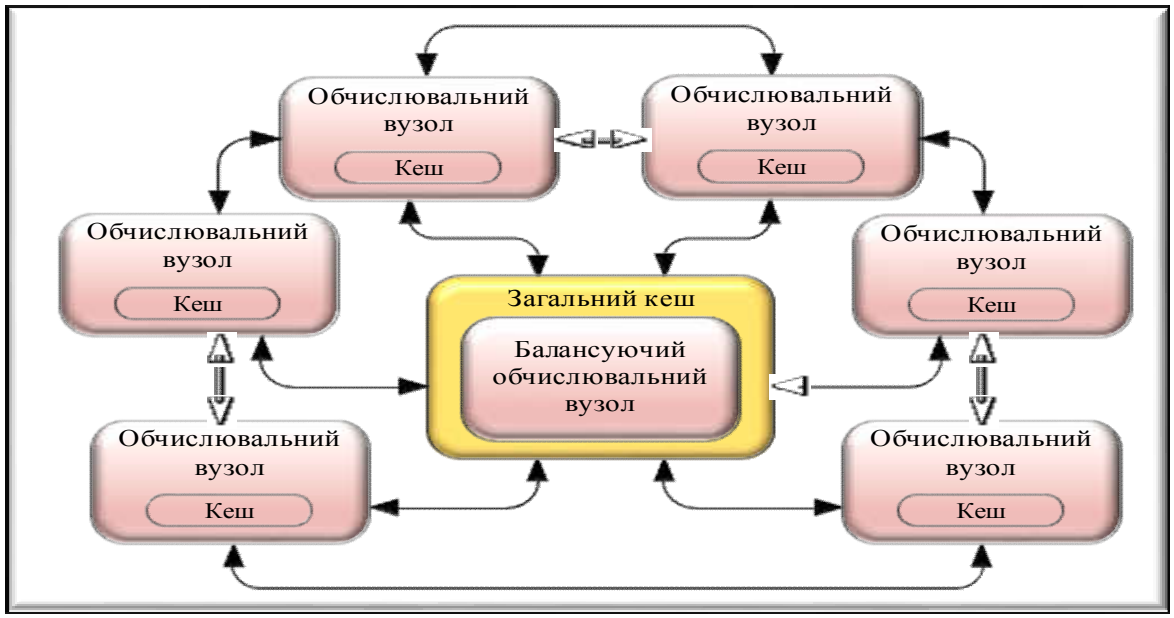


Рисунок 4.10 – Модель програмної Web–системи з балансувальним вузлом та загальним кешем

Методика відправлення інформаційних пакетів повинна використовувати алгоритм розрахунку контрольної суми протоколу управління передаванням пакета, який дає змогу відобразити контрольну суму протоколу управління передаванням даних пакета за допомогою контрольної суми заголовка цього пакета (рис. 4.11).

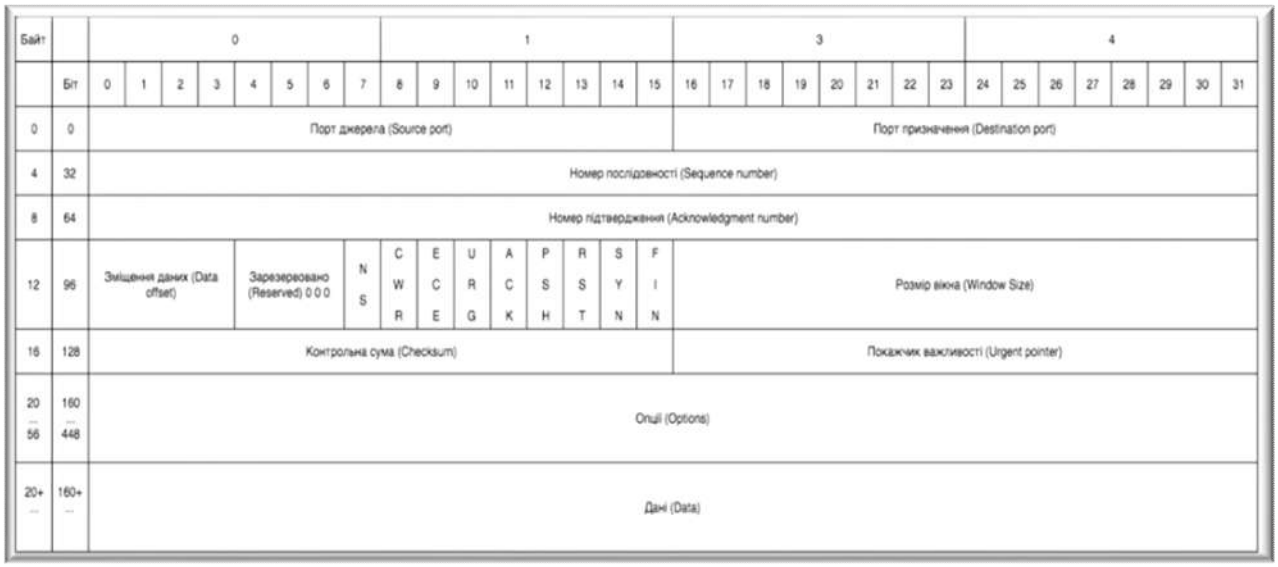


Рисунок 4.11 – Модель сегмента протоколу управління передаванням пакета даних

Розглянувши алгоритм перерахунку контрольної суми, який полягає у тому, щоб для підрахунку контрольної суми використовувати контрольну суму вихідного пакета та контрольну суму вхідного пакета. Отже, якщо буфер обміну розбити на дві інформаційні частини, то контрольну суму всього буфера можна лінійно виразити через контрольні суми його частин. Типова довжина заголовка протоколу управління передаванням пакета, зазвичай має меншу довжину у декілька разів, ніж основа всього інформаційного пакета. Отже, зменшується обчислювальне навантаження в n-разів ресурсів усієї програмної Web-системи та дає змогу значно зменшити вартість переадресації клієнтського запиту з одного вузла на інший. На наступному технологічному етапі запит користувача переходить на обчислювальний вузол для опрацювання. Архітектура обчислювального вузла для опрацювання інформаційних запитів користувача має бути організована у стилі ієрархії. У низу цієї ієрархії неважливі обчислювальні модулі опрацювання користувацьких запитів, відмова яких може тимчасово сповільнити загальну роботу Web-системи, але не паралізувати її в загалом. Якщо нижні обчислювальні модулі не працюють,

то хмарна сіткова Web-система дає змогу використовувати відповідні обчислювальні модулі інших вузлів. Однак при такій організації обчислювального процесу зрозуміло, що рівень комунікації між вузлами з опрацювання користувацьких запитів має бути досконалим.

Особливості організації архітектури обчислювального вузла з опрацювання користувацьких запитів вимагають чіткої ієрархії цих модулів. Її суть полягає у тому, що кожен із обчислювальних модулів вузла повинен бути максимально ізольованим від інших, а проводити обмін інформацією лише на рівні загальних системних повідомлень, водночас будучи інформаційно залежним. Отже, проєктування програмної Web-системи необхідно розробляти за таким принципом: «де збій – це певна частина роботи Web-сервісу, яка має бути контрольована на рівні обміну системними повідомленнями». Архітектура ієрархії відмов є в тому, що будь-яка Web-система складається з обчислювальних модулів, а наступний володіє послідовністю дій тоді, коли відмовили модулі нащадків. Інформаційний потік даних у такому випадку йде по висхідному порядку, бо лише батьківський обчислювальний модуль містить дані про те, які саме дії, необхідно виконати, якщо один або декілька модулів не працюють на хмарній обчислювальній платформі. Проте тоді, коли один із обчислювальних модулів вузла вийшов із ладу або завантажений, то здійснюється запит до іншого модуля, вищого у своїй ієрархії. Також необхідно зауважити, що основний обчислювальний модуль може і не брати участь в опрацюванні клієнтського запиту, а виступати, як Web-сервіс, який може перебувати у безпосередньому контакті з іншими вузлами хмарної мережі та паралельно проводити діагностику відповідних модулів вузла. Після цього опрацьовуються інформаційні потоки даних, згенерованих користувачем, а якщо її неможливо виконати, то приймається управлінське рішення переправити інформаційний потік від непрацюючого або сильно завантаженого модуля, до такого у якого є надлишкові ресурси (рис. 4.12).

Вибір обчислювального модуля для виконання запитів користувачів ґрунтується не лише на його завантаженості, а і на інформації про кількість збоїв в інших модулях хмарного вузла. Такий алгоритм необхідний для того, щоб не створювати інформаційні пакети для миттєвого перебігу даних для опрацювання від вкрай завантажених модулів вузла до менш завантажених. Внаслідок цих маніпуляцій утворюється складна хмарна мережа обміну масивами обчислень між відповідними модулями обчислювальних вузлів. Однак головною ме-

тою при побудові високонавантажених розподілених програмних Web-систем є створення не такої програмної системи з використанням хмарної технології яка не вийде із ладу під час промислового використання, а тієї, що максимально довго зможе працювати і справлятися з різноманітними інформаційними викликами та власними помилками. Після того, як потік інформаційних повідомлень був переадресований, обчислювальний модуль не припиняє своєї роботи. Отже, можна акцентувати стратегії, які можна втілити у кожному із цих обчислювальних модулів.

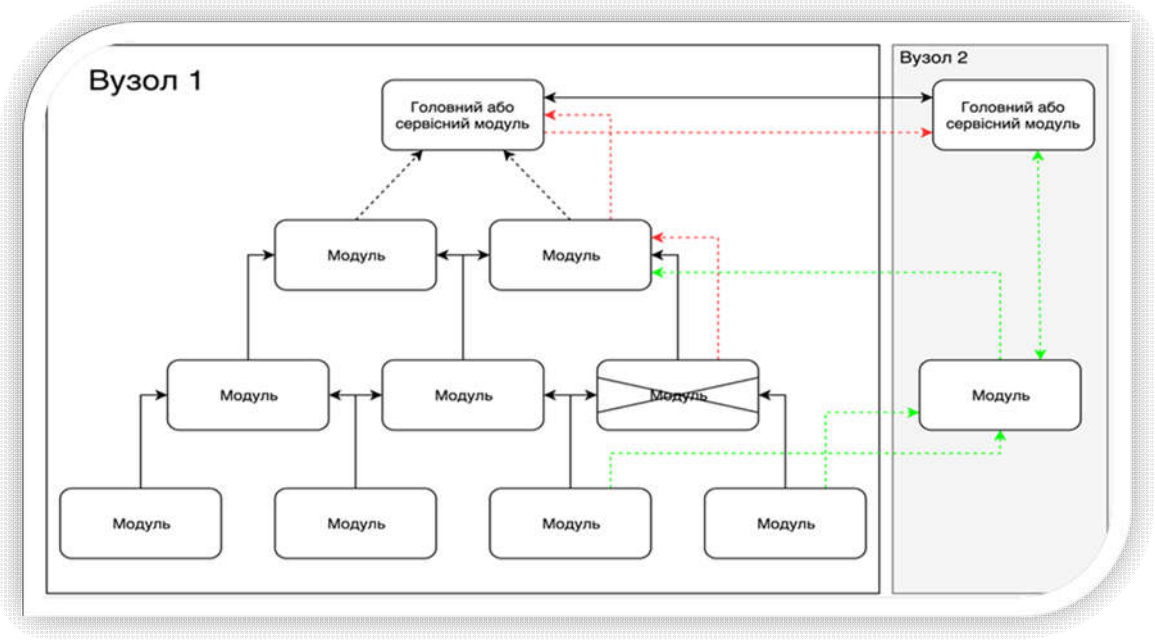


Рисунок 4.12 – Модель опрацювання ієрархії відмов у програмних Web-системах

Під першою стратегією будемо розуміти що, обчислювальний модуль знаходився під навантаженням. У такій ситуації модуль виконує весь список користувацьких завдань, що надійшли до нього та залишилися в ньому для опрацювання, здійснюючи первинне перезавантаження, та формує інформаційне повідомлення модулю, який перебуває вище в ієрархії і готовий до роботи. З огляду на те, що хмарна сітка модулів обчислювальних вузлів та основний балансувальник навантаження постійно обмінюються інформаційними повідомленнями про готовність до опрацювання, а модуль отримає новий потік даних для обробки при кожній наступній ітерації надходження даних в розподілі навантаження на вузлах.

Метод дублювання даних snapshot/restore часто застосовується у високонавантажених розподілених Web-системах. Отже, модель гіб-

ридної системи з центральним балансувальним вузлом розподілу навантаження та хмарною сітковою комунікацією між обчислювальними вузлами дає змогу достатньо просто додавати все нові вузли завдяки застосуванню кластерної структури. Високий рівень комунікації обчислювальних вузлів між собою створює проблему для методу клонування даних, оскільки у випадку відмови одного із вузлів необхідно зробити перерахунок показника інтегральної продуктивності Web-системи загалом. Зрозуміло, що у такому випадку можна розділити виниклі проблеми, які пов'язані з відмовою обчислювального вузла на дві умовні групи. Перша група проблем програмного забезпечення та апаратно технічних. Оскільки перерозподіл обсягу обчислень відбувається на рівні обчислювальних модулів тоді, якщо програмне забезпечення розробленої Web-системи згенерує помилку, то система автоматично та швидко відновить працездатність, якщо її обчислювальний модуль на вершині ієрархії був належно спроєктований та не набув каскадної помилки. Тоді він сповістить інші обчислювальні вузли про наявність помилок та перенаправить потік даних обчислень на інші модулі. Отже, у цій ситуації не відбувається дублювання конкретного обчислювального вузла на незалежній платформі, а просто виконується запуск типового екземпляра.

Для того щоб суттєво зменшити ризик втрат значної кількості запитів користувачів, необхідно, щоб балансувальний обчислювальний вузол містив буфер, що буде виступати як «повторна спроба» у тому випадку, коли фізично недоступний певний вузол. Тому використання методу дублювання, забезпечує створення типового обчислювального вузла і водночас відновлює продуктивність Web-системи. Проте у розробці програмних систем з використанням хмарної технології існує цілий клас розподілених алгоритмів, де є визначені структури інформаційних потоків даних (масиви, записи), розмір яких може залежати від загальної кількості одночасно взаємодіючих процесів, що виконують опрацювання різними обчислювальними вузлами на незалежних платформах. Прикладом застосування такого алгоритму може слугувати протокол прийняття узгодженого рішення. При модифікації топології розподіленої високонавантажувальної обчислювальної Web-системи, також буде змінюватись структура потоків даних та алгоритмів їхнього опрацювання. Отже, виникає нагальна необхідність у модифікації сценарію поведінки використаної імітаційної моделі, а саме у зміні алгоритмів та структур потоків даних, які описують поведінку відповідних об'єктів.

4.6. Реалізація методів опрацювання потоків даних для відновлення обчислювальних вузлів

При розробці високонавантажувальних розподілених Web-систем, які архітектори складають з декількох обчислювальних вузлів, та які є екземпляром єдиної обчислювальної системи, таким чином утворюючи «кластеризацію». Однак модифікований алгоритм опрацювання потоків даних потребує чіткого поділу не лише на фізичному рівні, а й на програмному. Програмне забезпечення Web-систем традиційно ділиться на логічні модулі, тобто за функціональним призначенням. При розробці програмної Web-системи архітектору необхідно проєктувати її так, щоб обчислювальні модулі у своїй структурі мали якомога менше логіки, оскільки головним завданням перерозподілу обсягу навантажень є підтримка системної рівноваги [95, 211]. Не менш важливою вимогою до архітектури програмного забезпечення Web-системи є уникнення сильної зв'язності. Сильна зв'язність між обчислювальними модулями полягає у тому, що один і модуль вступає у взаємодію з значною кількістю інших модулів на незалежних платформах. Тобто, якщо у разі його поломки значна кількість модулів надішлють повідомлення у сервісний модуль, а той в свою чергу сповістить інші вузли про необхідність їх тимчасової заміни. Щоб ефективно подолати можливі обчислювальні проблеми, можна використати такі методи відновлення працездатності розробленої програмної Web-системи з використанням хмарної технології:

Метод clone_args зберігає значення аргументів, які були відправлені обчислювальному модулю, що вийшов із ладу. Оскільки відсутність своєчасної відповіді від модуля класифікується, як його відмова у такому випадку батьківський метод модуля має отримати відповідну інформацію про аргументи, які були надіслані дочірньому методу. Однак на цьому прямі особливості методу *clone_args* не закінчуються, зокрема він має опосередковані властивості, до яких належить:

- поточний інтегральний показник часу опрацювання для обчислювального модуля або методу модуля, що відмовив;
- фіксований час, за який відбулася відмова опрацювання обчислювальних вузлів хмарних Web-систем;
- частота запусків обчислювального модуля, яка дає змогу визначити потребу в цьому модулі, а отже, під час розподілення наван-

таження між вузлами потрібно більш ефективно виділяти необхідні програмно-апаратні ресурси.

Метод report дає змогу відправляти з певними інтервалами часу статистичні дані для сервісного модуля про опрацювання потоків даних усіх дочірніх. Також його використання сприяє автоматизованому прогнозуванню динаміки деградації розробленої програмної Web-системи. Отже, метод *report* можна застосовувати не лише для того, щоб інформувати про планові статистичні показники, а й тоді, коли сталася відмова одного або декількох дочірніх обчислювальних модулів.

Метод restart забезпечує розроблену програму Web-системи з використанням хмарної технології перезавантаженням обчислювального модуля у тому випадку, коли пройшла його відмова. Основна мета перезавантаження – уникнення руйнівних вхідних потоків даних.

Метод callback дає змогу надіслати інформаційне повідомлення батьківському модулю про успішне або неуспішне перезавантаження обчислювального вузла. Цей метод передається як аргумент обчислювального модуля, який необхідно перевірити щодо стабільної роботи після запуску або перезапуску програмної Web-системи, що забезпечує «метод зворотного виклику». Проаналізувавши характер помилок, які можуть виникнути в обчислювальних модулях розробленої програмної Web-системи, зауважимо декілька основних причин: помилки у програмному коді, проблемні потоки даних та високе навантаження у вузлах.

Метод feedback забезпечує зв'язок між обчислювальними вузлами розробленої програмної Web-системи з використанням хмарної технології та є максимально незалежним від інших, тому його може оточувати багато мікросервісів Web-системи, з якими необхідно обмінюватися потоками інформації. Для цього, звичайно, потрібно повідомити сервісний обчислювальний модуль, який перебуває у горі вершини відповідної ієрархії. Тому кожен обчислювальний модуль реалізує метод *feedback*, який знає, як зв’язатися з ним, та здатний перерозподілити потоки даних від модулів, які вийшли із ладу або сильно завантажені, до робочих вузлів. Модифікована обчислювальна модель методики відновлення, модулів які вийшли із ладу, подано на рис. 4.13.

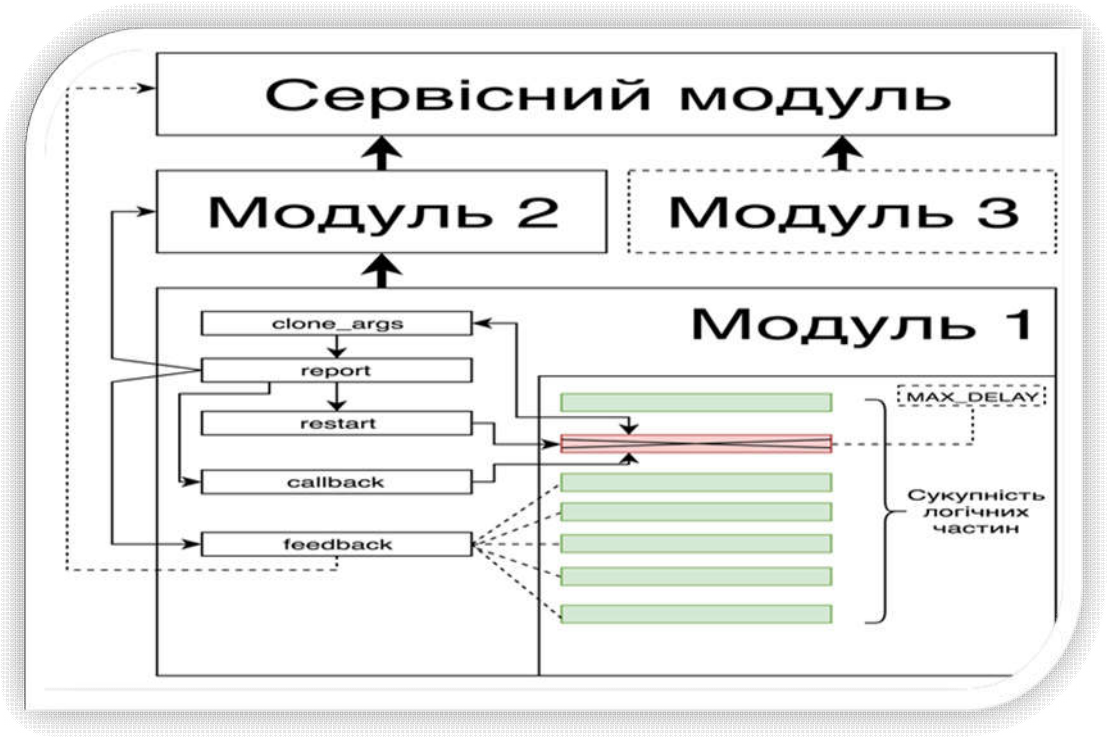


Рисунок 4.13 – Алгоритм перерозподілу навантаження при відмові обчислювальних вузлів програмної Web-системи

Цей модифікований алгоритм реалізують лише сервісні модулі з опрацювання інформаційних потоків даних. На відміну від звичайних обчислювальних модулів, сервісні модулі забезпечують виконання значно меншої кількості операцій при отриманні позитивного результату, який безпосередньо впливає на їхню роботу. У такому випадку сервісні модулі виступають «стійкою частиною програмної системи в одній обчислювальній точці» щодо вузла на незалежній обчислювальній платформі. Отже, при розробці програмної Web-системи важливо використовувати розглянуті методи для забезпечення мінімальної вразливості даного обчислювального модуля. Ці обчислювальні методи постійно інформують про результати роботи через певний квант часу або позапланово у випадку виходу із робочого стану. Зрозуміло, що сервісний модуль має інформацію про роботу всієї розробленої програмної Web-системи [219]. Чим менший квант часу оновлення інформаційних потоків даних про результати роботи розробленої програмної Web-системи, тим точніше та ефективніше можна визначити необхідний обчислювальний вузол для перенесення його навантаження.

4.7. Динамічні процеси перенесення обчислювальних навантажень між вузлами

Існує дві стратегії визначення потенційного обчислювального вузла, який приймає інформаційний запит для опрацювання. Сервісний модуль опрацювання користувацьких запитів на базі власних потоків даних визначає список вузлів претендентів для перенесення навантаження. Згодом відправляє користувацький запит, який містить інформацію про вузол, що вийшов із робочого стану та загальну його позицію у списку запропонованих вузлів для наступного опрацювання потоків даних. Отже, до обчислювального вузла може водночас надходити певна кількість запитів від n – кількості вузлів. Цей підхід в опрацюванні інформаційних потоків даних може створювати суттєві затримки у роботі з особливо трудомісткими завданнями. Тому, на нашу думку, варто використовувати іншу стратегію опрацювання потоків даних, де кожен обчислювальний вузол на основі отриманих даних інформує інші подібні вузли про готовність збільшення обсягу навантаження для опрацьовування збільшених потоків інформації. Один або декілька вузлів можуть претендувати на додаткові обчислювальні ресурси. Коли вузол відправляє запит на отримання ресурсів, про це стає відомо іншим вузлам, а вузол, який отримує інформацію, закриває позицію про готовність приймати додаткові обчислення від інших. Прикладом використання цієї системи є декілька одночасних черг до продавців, коли кожен із клієнтів бачить рівень їхнього поточного завантаження продавців, а тому зразу намагається у найменш завантажену чергу для опрацювання. Оскільки ця черга не прихована, то простий підрахунок клієнтів у черзі дає змогу швидко зараз визначити оптимального продавця. Проте Web-системі недостатньо враховувати можливості та потужності обчислювальних серверів для опрацювання потоків даних, а також вкрай важливо враховувати і завантаженість інформаційних каналів.

Ця проблема практично не виникає на обчислювальних вузлах, які територіально розташовуються у межах однієї будівлі, але здебільшого використання інформаційних систем, обчислювальні вузли на незалежних обчислювальних платформах знаходяться на значних відстанях, тому важливо врахувати вплив значних потоків даних на навантаження вузлів та комунікаційних каналів. Ступінь завантаження обчислювального каналу передачі потоків даних визначається часткою інтегрального часу роботи програмної системи, протягом якого у

ньому здійснювалась передача даних. Щоб визначити загальний показник завантажуваності каналу, а саме чи достатньо сумарного обсягу пропускної здатності каналу для передачі потоків даних при поточній величині навантаження, насамперед необхідно провести розрахунок ступеня завантаження цього каналу у фіксований період часу (t_0; T), коли загальна кількість користувацьких звернень до Web-системи максимальна. Отже, вихідними потоками даних є інформація про час звернення до об'єктів та обсяг отриманих даних. Для проведення коректних обчислень необхідно визначити такі припущення:

- уся пропускна здатність інформаційного каналу V, що вимірюється у біт/с, що виділена лише для передання потоку даних Web-системи відома;

- передача інформаційного потоку даних починається одразу, якщо у момент їх надходження канал простоює (вільний). В іншому випадку інформаційний потік даних додається у сформовану чергу на передання.

Нехай за певний період спостережень (t_0; T) зафіксовано N звернень до обчислювальних обсяги, а також відомі кванти часу звернень об'єктів $t_1, t_2, ..., t_N$ та об’єми переданих потоків даних на опрацювання $d_1, d_2, ..., d_N$. Отже, динаміку надходження інформаційних потоків даних до входу каналу даних для їх передачі можна зобразити у вигляді діаграми, а відповідно її динаміку зміни розміру черги w обчислювальних даних для передачі.

У початковому кванті часу t_0 розмір черги інформаційного потоку даних $w=0$. У квант часу t_1 надходить інформаційний потік даних від першого об'єкта з обсягом d_1, водночас розмір черги набуває значення $w_1=d_1$, а оскільки канал для передачі даних був вільний, то дані починають негайно надходити зі швидкістю V, а розмір черги, відповідно, зменшується (рис. 4.14).

Для передачі інформаційного потоку даних обсягом w_1 необхідно витратити час t'-t, де t' – час закінчення передання даних по інформаційному каналу від першого об'єкта. Загалом час, необхідний для передання певного обсягу інформаційного потоку даних w_i, які перебували у черзі на момент часу t_i, можна обчислити за формулою (4.24):

$$t_i' = t_i + \frac{w_i}{V}, \tag{4.24}$$

де t' – час закінчення передання інформаційного потоку даних від першого об'єкта; V – швидкість передання об’єкта; w_i – загальний об-

сяг інформаційного потоку даних для передачі; t_i – момент надходження інформаційного потоку даних.

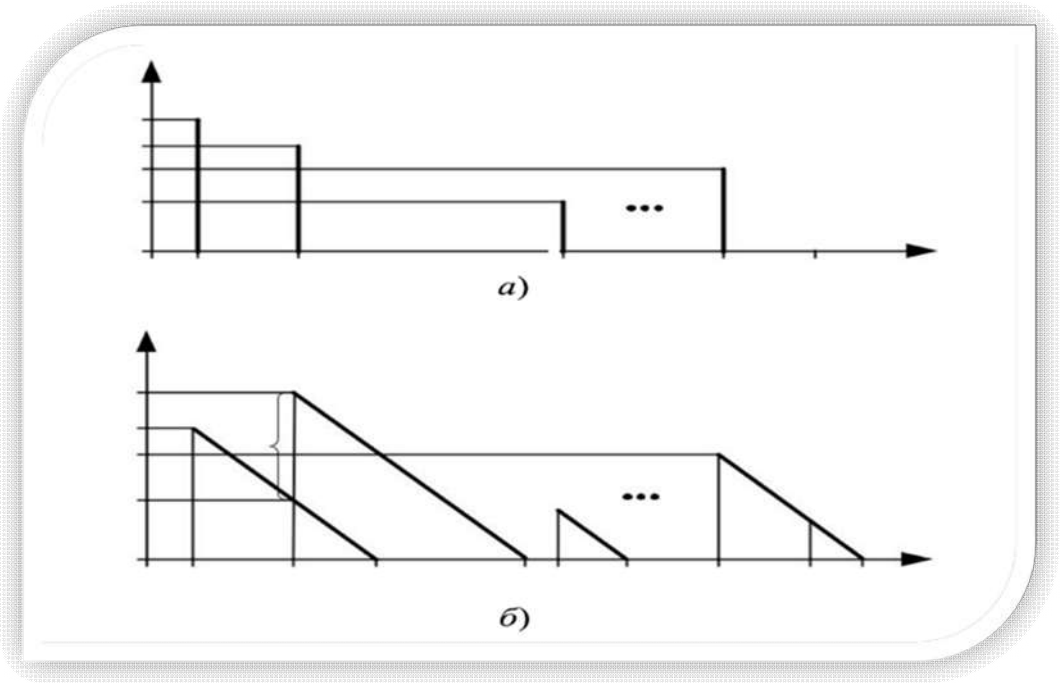


Рисунок 4.14 – Динаміка зміни черги інформаційного потоку даних для передачі

У момент часу t_2, коли надійшов інформаційний потік даних від другого запиту обсягом d_2, а передача даних по каналах зв'язку від попереднього об'єкта ще не завершилася, то розмір черги збільшиться і набуде сумарного значення $w_2=w_{залиш1}+d_2$, де $w_{залиш1}$ – розмір сформованої черги потоку даних для передачі на наступний момент часу t_2. Загальна передача потоку даних обсягом w_2 відбудеться у момент часу t'_2. Якщо до настання моменту часу t_3 нові інформаційні потоки даних не надходять у канал зв'язку, то він набуває статусу вільного. Отже, рівень завантаження інформаційного каналу зв'язку у кванти часу (t_0; T) можна розрахувати за формулою (4.24):

$$p = \frac{t_{\text{перед.}}}{t_{\text{загальн.}}} = \frac{d_\Sigma}{Vt_{\text{загальн.}}}, \qquad (4.24)$$

де d_Σ – сумарний обсяг інформаційного потоку даних за період спостережень; $t_{перед.}$ – час, протягом якого канал зв'язку був зайнятий передачею потоку даних, запитаних за період (t_0; T); $t_{загальн.}$ – сумарний час роботи каналу зв'язку, що визначається довжиною періоду (t_0; T),

якщо передача інформаційного потоку даних завершиться до моменту часу T.

У випадку, якщо передача інформаційного потоку даних завершиться пізніше, ніж визначений момент часу T, то необхідно враховувати це для наступного розрахунку, оскільки за умовою (4.25) $t_{перед.}$ розраховується з огляду на те, що усі потрібні дані інформаційного потоку передані, отже, $t_{загальн.}$ розраховується згідно з таким виразом:

$$t_{загальн.} = \begin{cases} T - t_0, t'_N \leq T \\ t'_N - t_0, t'_N > T \end{cases}, \tag{4.25}$$

де $t_{загальн.}$ – сумарний час роботи каналу зв'язку з переданням потоків даних; t' – час закінчення передачі інформаційного потоку даних від об'єкта, що надійшли за період (t_0; T); T – момент часу.

З огляду на алгоритм обміну інформаційними потоками даних між обчислювальними вузлами можемо визначити загальний рівень завантаженості каналу зв'язку для передачі потоків даних за допомогою математичного виразу (4.26):

$$p = \begin{cases} \dfrac{d_\Sigma}{V(T - t_0)}, V \geq \dfrac{w_N}{T - t_N} \\ \dfrac{d_\Sigma}{w_N + V(t_n - t_o)}, V < \dfrac{w_N}{T - t_N} \end{cases}, \tag{4.26}$$

де p – загальний рівень завантаження каналу зв'язку для передачі потоків даних; d_Σ – сумарний обсяг переданих потоків даних за період спостережень; w_n – обсяг потоку даних; t_N – час завершення передачі потоку даних; V – швидкість передачі; T – момент часу.

Математичний вираз (4.26) дає змогу розрахувати поточний рівень завантаження каналу зв'язку для передачі потоків даних на підставі їхнього моніторингу. Поставивши перед собою за необхідним рівнем керування завантаження ρ каналу зв'язку для передачі потоків даних та використавши метод дихотомії, можемо обчислити загальну величину пропускної здатності каналу зв'язку для передачі потоків даних за допомогою формули (4.26). Вихідною величиною для розрахунку є обсяг пропускної здатності каналу зв'язку, який необхідний для передачі потоків даних, що опрацьовані за аналізований період. Основна суть цього методу полягає у тому, що в його основі лежить моніторинг аналізу інформаційних потоків даних про поточне навантаження каналів зв'язку, яке накопичується у розробленій програмній Web-системі з використанням хмарної технології, що дає змогу сут-

тєво спростити математичні розрахунки, а також оперативно отримувати актуальні показники характеристик каналу зв’язку для передачі інформаційних потоків даних за будь-яких значних змін їхніх обсягів.

Пакування, перенесення станів та інформаційних потоків даних

Як тільки у розробленій програмній Web-системі з використанням хмарної технології між обчислювальними вузлами було досягнуто консенсусу, які делегують та приймають частину обчислювальної роботи, необхідно підготувати пакет неопрацьованих даних новому вузлу. А лише після цього встановити зв'язок між модулями, що надсилають користувацькі запити до тих, які вийшли з ладу, та на інший робочий модуль в іншому обчислювальному вузлі. Також сервісний модуль посилає повідомлення з позначкою, що є пошкоджений обчислювальний модуль – «тимчасово недоступний для нових запитів на опрацювання потоків даних», потім виконує стандартну процедуру їхнього перезавантаження. Ця процедура реалізована у кожному обчислювальному модулі завдяки відповідному методу *restart,* також вона пов’язана із сервісним модулем на обчислювальному вузлі методом *feedback.* Як тільки розроблена програмна система отримує повідомлення, що обчислювальний модуль вийшов із ладу, *clone_args* то відбувається копіювання усі вхідних потоків дані, які були надіслані на опрацювання, та опосередкований набір даних про актуальний стан системи.

У такому випадку часу на прийняття управлінських рішень обмаль, тому сервісний модуль якомога швидше намагається дібрати новий обчислювальний вузол, водночас копіюючи увесь потік інформаційних викликів через метод *clone_args* та надсилає їх у сервісний модуль. При аналізі послідовності відповідних дій стає зрозуміло, що необхідно буде формувати список викликів на опрацювання, який зараз чекають інші батьківські обчислювальні модулі або їхні методи. Проте цей список і є своєрідною чергою на опрацювання запитів користувача, які має опрацьовувати новий робочий обчислювальний модуль за створеною чергою. За аналогічним принципом працює Amazon Simple Queue Service – сервіс, який оперативно приймає черги користувацьких повідомлень для зберігання та опрацювання. Як тільки обчислювальний вузол стає доступним для інформаційних потоків даних, сервісний вузол компонує ці дані та зв’язує відповідні модулі і їхні стани (рис. 4.15).

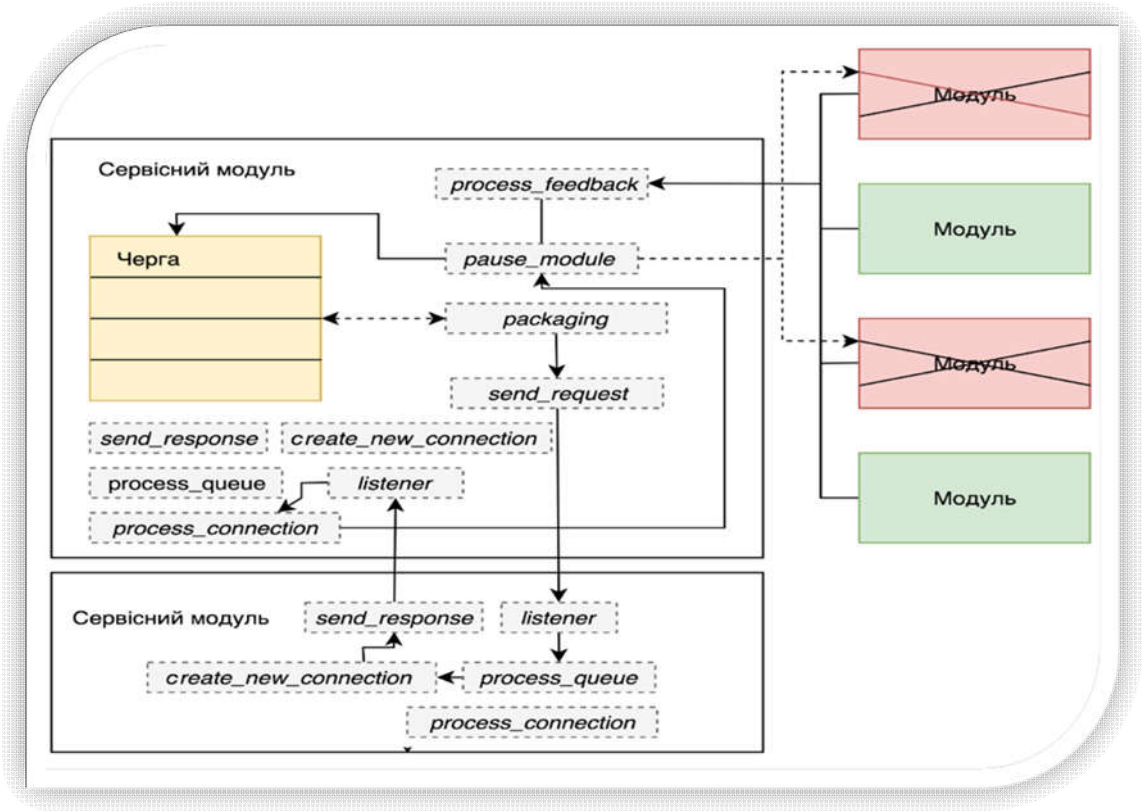


Рисунок 4.15 – Модель пакування, перенесення станів та інформаційних потоків даних

Для забезпечення надійності передачі інформаційних потоків даних необхідно провести розділення конкретної сукупності цих даних на відповідні фрагменти та додати до кожного із них заголовок із номером послідовності. Отримані у такий спосіб фрагменти інформаційних потоків даних формують сегмент. На наступному кроці опрацювання даних кожний сегмент переходить у пакет, а потім за допомогою транспортних інформаційних протоколів надходить до обчислювального вузла отримувача.

Після доставлення пакета до обчислювального вузла отримувача перевіряється коректність отримання інформаційного потоку даних у сегменті за допомогою перерахунку контрольної суми, та автоматично встановлюється, що попередні сегменти інформаційних даних також успішно отримані. На цьому етапі обчислювальний вузол отримувача надсилає інформаційний запит до вузла відправника про новий пакет даних або повторна передача цього пакета даних. Вказаний алгоритм операцій забезпечує підтвердження того, що всі попередні пакети із послідовності потоку даних були успішно отримані.

У розробленій модифікованій моделі обчислювальні модулі ізольовані один від одного і не мають інформації про загальний стан

програмної Web-системи, але, встановивши зв'язок між ними, можна отримувати інформаційні повідомлення про їхні стани. Для взаємодії обчислювальних модулів використаємо асинхронний обмін інформаційними повідомленнями, де кожен відповідний модуль має свою чергу інформаційних повідомлень. Отже, обчислювальний модуль, який відправив повідомлення, очікує наступного сповіщення про його доставку, але тоді, коли отримувач ігнорує це повідомлення.

Висновки до четвертого розділу

У розділі проведено системний аналіз технологій та концепцій, які застосовуються для розробки високонавантажувальних розподілених відмовостійких програмних Web-систем з використанням хмарної технології. Проведений системний аналіз розглядає базові принципи дизайну розподілених Web-систем та технології, які найчастіше використовують архітектори програмних систем при проєктуванні. Також ми дійшли висновку, що резервування – неодмінний атрибут при розробці більшості Web-систем, що є високо навантажувальними відносно користувацьких запитів. Основним критерієм при розробці програмних Web-систем є їхнє масштабування, яке застосовують тоді, коли операції потребують чималих обчислювальних ресурсів, що значно знижує продуктивність системи та вимагає від неї загального збільшення потужності. Розглянуті особливості розробки програмних Web-систем із вертикальним та горизонтальним масштабуванням.

Також у розділі досліджено методи оцінки надійності і відмовостійкості розроблених програмних Web-систем, тому що будь-яку програмну систему треба об’єктивно моніторити, та вона має бути прогнозованою у своїй роботі. Розглянуто основні аспекти експлуатації розподілених відмовостійких програмних Web-систем, оскільки проблеми їхнього адміністрування, а також відновлення працездатності при виході із робочого ладу можуть виникати у процесі високонавантажувальної експлуатації.

Аналізуючи технології, що використовуються при розробці програмних Web-систем, у дослідженні ми удосконалили методи, згідно з якими проведено комплексний аналіз балансування обчислювального навантаження, яке є основним завданням при проєктуванні програмних високонавантажувальних розподілених відмовостійких Web-систем з використанням хмарної технології. Для вирішення питання ефективної роботи розробленої програмної системи застосовуються

алгебраїчні методи оптимізації роботи балансування обчислювального навантаження на вузлах, а також мережева модель його розподілення, тим самим створюючи гібридний механізм перенесення інформаційних потоків даних.

Розглянуто теоретичні та практичні основи, що забезпечують ефективне функціонування механізму ієрархії відмов, які дають змогу забезпечити ефективний розрахунок обчислювального навантаження програмних модулів та вузлів у загалом. Ґрунтовний аналіз досліджень показав, що потужність промислових комп'ютерів обмежена, а їхня вартість зростає значно швидше, ніж обчислювальна продуктивність, отже вартість у два рази потужнішого комп'ютера зросте понад два рази. Аналізуючи високонавантажену розподілену відмовостійку програмної Web-систему, встановили, що основоположним її критерієм є ефективність опрацювання потоків даних, яка загалом випадку обчислюється як частка від ділення суми усіх вихідних об’ємів параметрів на суму всіх вхідних потоків даних. Для кожного конкретного обчислювального модуля чи вузла визначається власна величина ефективності. Порівняння ефективності обчислювальних вузлів відбувається з використанням методу лінійного програмування та водночас різних базисних моделей та їхніх варіантів. Запропоновано визначати кількість залучених обчислювальних модулів або вузлів шляхом побудови критерію межі ефективності, а для всіх інших випадків – критерію міри їх неефективності.

Удосконалена функціональність сіткової моделі, яка значно відрізняється від звичайних послідовних та паралельних її моделей. Основна відмінність – можливість агрегування та спільного використання значних наборів гетерогенних обчислювальних ресурсів з опрацювання інформаційних потоків даних, розподілених географічно. У багатьох випадках це приносить значні переваги, оскільки розроблена програмна Web-система з використанням хмарної технології вимагає додаткових ресурсів, які недоступні у межах одного обчислювального вузла, водночас їх можна отримати з інших вузлів, які під’єднані до функціональної сітки.

У дослідженні удосконалено алгоритм перенесення обчислювального навантаження без застосування резервних ресурсів та запропоновано модифіковану модель організації взаємодії обчислювальних модулів з опрацювання інформаційних потоків даних у межах одного вузла, а також розробленої програмної Web-системи загалом.

У розділі розглянуто методику визначення причин виходу із ладу обчислювальних модулів та вузлів, а також акцентовано два типи основних проблем, які пов’язані із виходом вузла з ладу, проблеми функціонування програмного забезпечення та апаратно-технічні.

Модифіковано модель управління обчислювальним процесом з опрацювання інформаційних потоків даних, де ізольовані один від одного модулі не мають загального стану, проте між ними можна встановити інформаційний зв'язок та отримувати повідомлення про їхні стани. Для взаємодії обчислювальних модулів використано асинхронний обмін інформаційними повідомленнями, де кожен організовує свою чергу повідомлень. Модуль, що відправив інформаційне повідомлення, очікує підтвердження повідомлення про відповідну доставку, але не тоді, коли одержувач повідомлення його проігнорував.

Важливим аспектом, що розглядається у дослідженні, є те, що вільні обчислювальні ресурси розробленої програмної Web-системи з використанням хмарної технології можна знайти і в межах своїх можливостей. Приймаючи управлінське рішення про об’єктивний стан розробленої програмної Web-системи, балансувальний вузол аналізує кожний обчислювальний модуль зокрема, що не дає змоги йому об’єктивно оцінити завантаженість системи загалом. Проте сам обчислювальний вузол має актуальну та детальну інформацію про загальну роботу усіх його підсистем, що дозволяє йому ефективно та об’єктивно оцінити власні можливості.

РОЗДІЛ 5

ВИКОРИСТАННЯ ХМАРНОЇ ТЕХНОЛОГІЇ ДЛЯ РОЗРОБЛЕННЯ ПРОГРАМНИХ СИСТЕМ

5.1. Використання хмарної платформи як послуги

В умовах стрімкого розвитку інформаційно-комунікаційних технологій розширення безкоштовних хмарних сервісів стає актуальним застосування та впровадження у промислове використання новітніх інформаційних сервісів на основі спільного користування конкурентними технологіями. З огляду на це, з’явилась актуальна потреба в аутсорсингу інформаційно-комунікативних технологій – хмарних послуг (ІТ-послуги) у розробці програмних систем. Поняття «аутсорсинг ІТ» забезпечує передачу компанією будь-якого ІТ-процесу (програми, функції, роботи) або певної частини сторонній організації, що забезпечує професійні ІТ-послуги на незалежній обчислювальній платформі, це підтримка функціонування інформаційно-довідкових, експертних систем, забезпечення інформаційної безпеки баз та банків даних підприємств, зберігання й опрацювання значних обсягів даних, надання апаратних ресурсів. Аутсорсинг вирішує питання скорочення фінансових і часових витрат на впровадження, підтримку й модернізацію ІТ-інфраструктури. Він забезпечує конвергенцією інформаційно-комунікаційних середовищ, а саме зближення різноманітних електронних технологій підвищення бізнес-вимог до стабільності і доступності ІТ-послуг.

Для розробки програмної системи з використанням хмарних технологій команда розробників повинна розробити структурну схему, яка визначає основні функціональні властивості програмної системи, їхні взаємозв'язки та призначення. Під функціональністю програмної системи розуміють складові частини системи: елементи, пристрої, функціональні групи, функціональні ланки. Побудова структурної схеми, яка призначена для відображення загальної структури проєкту (поставленого завдання), – це розробка його основних блоків, вузлів, частин та формування головних зв'язків між ними. Зі структурної схеми програмної розробки системи з використанням хмарних технологій повинно бути зрозуміло, як система працює в основних режи-

мах роботи та як взаємодіють її частини. Позначення елементів структурної схеми можуть обиратись довільно, хоча загальноприйнятих правил виконання схем слід максимально дотримуватись [158, 304].

Хмарні технології (cloud computing) – це вільний спосіб доступу до зовнішніх інформаційно-комунікативних ресурсів у вигляді різноманітних Інтернет-сервісів. Термін «хмарні технології» був запропонований Рамнатом Челлаппатом (Ramnath Chellappa), який відзначив її як «обчислювальну парадигму, при якій межі обчислювальних елементів залежатимуть від економічної доцільності, а не тільки від технічних обмежень». Поява першої технології, надала доступ до додатків через сайт, а саме програмне забезпечення систем (ПЗС) як сервіс (Software as a Service [SaaS]).

Актуальність розробки розподілених програмних систем на базі хмарної технології полягає у тому, що потреба в сучасних високонавантажувальних системах невпинно зростає, а методів їхнього коректного розроблення та алгоритмів підтримки обчислювальної роботи на незалежних обчислювальних платформах у пікові моменти навантаження вкрай мало. У дослідженні акцентовано найбільш неоднозначні сучасні виклики у розробці програмних систем на базі хмарної технології, що стоїть перед архітекторами таких систем, а саме алгоритми переміщення навантаження між обчислювальними вузлами та архітектурами організації програмного забезпечення у межах бізнес-логіки підприємств. Використання запропонованої методики на незалежних обчислювальних платформ для розробки відмовостійких програмних систем на базі хмарної технології дає змогу розширити світоглядні підходи до побудови продуктивних механізмів розподілення обчислювального навантаження між вузлами відповідної системи [190, 307, 308]. З огляду на недостатнє дослідження цієї предметної області, ми розуміємо, що успішний алгоритм механізму перенесення обчислювального навантаження сприяє ефективній модифікації цього ресурсу у межах поставленого завдання, яке є головним критерієм у виборі програмних технологій архітектором програмних систем (рис. 5.1).

Ці концептуальні підходи та рішення мають недоліки та переваги. До його основної переваги належить простота адміністрування та балансування обчислювального навантаження між вузлами незалежних обчислювальних платформ розробленої програмної системи на базі хмарних технологій. А до недоліків цього підходу у розробці зараховують наявність додаткових вільних (не залучених) обчислюва-

льних ресурсів, створення додаткових дзеркал програмної системи для швидкого відновлення її працездатності.

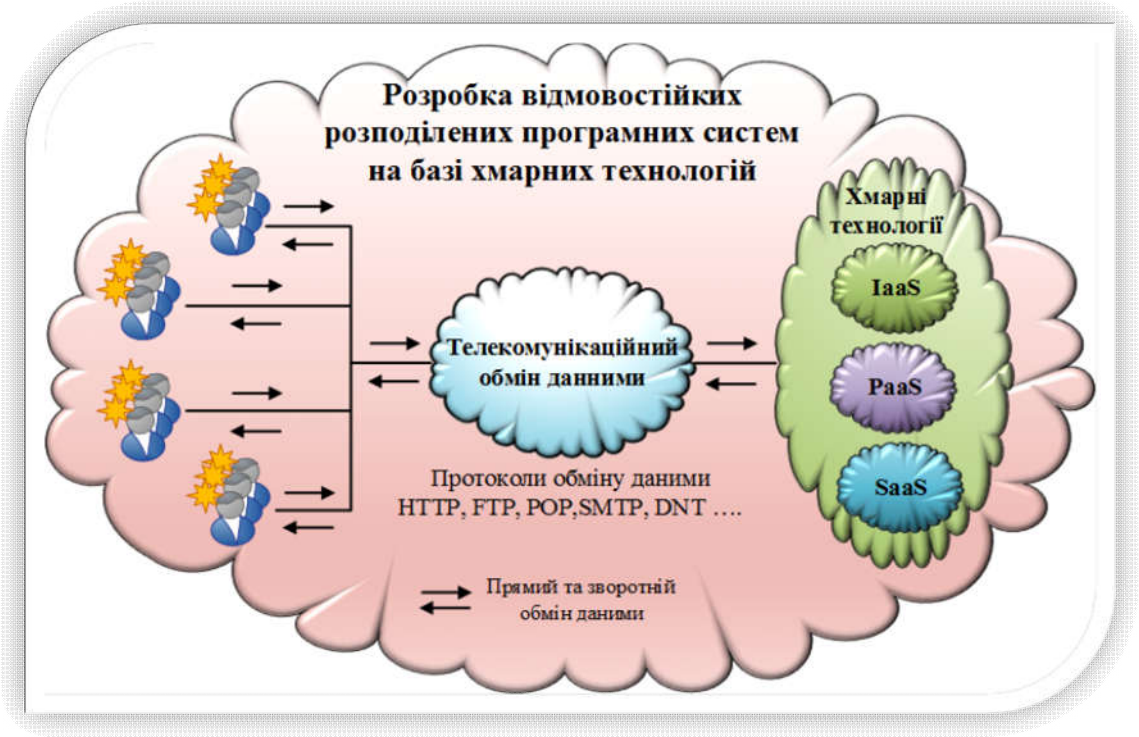


Рисунок 5.1– Модель відмовостійкої розподіленої програмної системи на основі хмарних технологій

Деякі дослідники передбачають використання майбутнього Інтернету, як вид «віртуального» суперкомп'ютера (незалежної обчислювальної платформи), яка буде залежати переважно від функцій та специфіки надання послуг хмарних обчислень, таких як об'єднання обчислювальних ресурсів та їхня віртуалізація, послуг на вимогу і незалежний доступ до програмної системи та даних із будь-якої точки світу у будь-який час.

Зараз існує чотири типи хмар:

1. Громадська хмара, яка надає послуги мешканцям землі.
2. Приватна хмара надає послуги для користувачів тільки у межах однієї організації або корпорації.
3. Хмара спільноти надає послуги конкретному товариству організацій та приватних осіб.
4. Гібридні хмари – це будь-яка комбінація перших трьох типів.

Як показано на рис 5.2, архітектура хмарних обчислень має три незалежні шари: програмне забезпечення обчислювальних систем як

послуга (SaaS), платформа як послуга (PaaS), а також інфраструктура як послуга (IaaS).

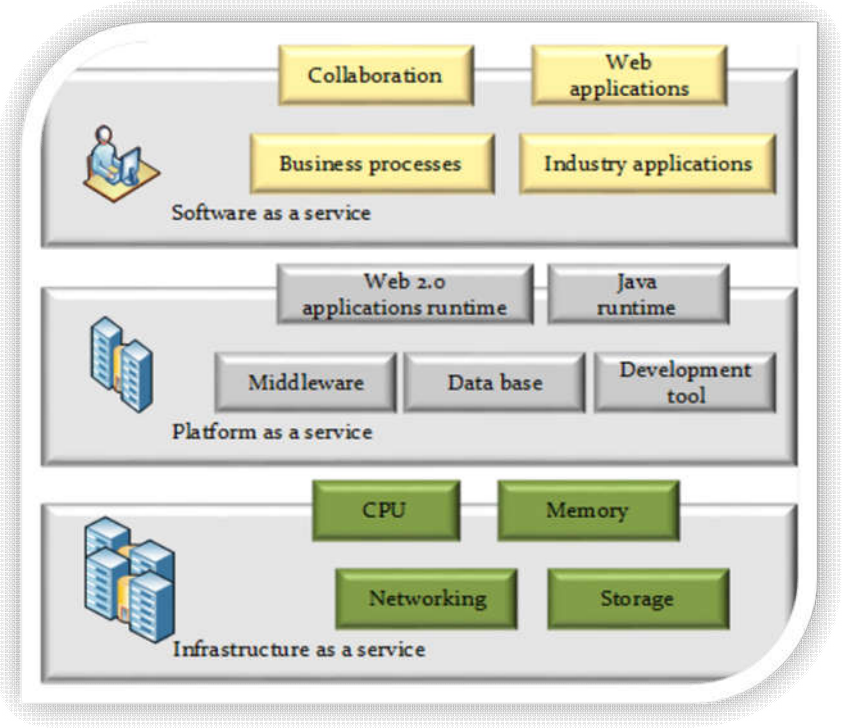


Рисунок 5.2 – Структура розподілених хмарних сервісів

Розробники високонавантажувальних розподілених програмних систем з використанням хмарних технологій можуть реалізовувати та використовувати кожен шар, як відповідну службу на незалежних обчислювальних платформах. Структура надання хмарних послуг на незалежних обчислювальних платформах для розробки програмних систем з використанням хмарних технологій відображена на рис. 5.3.

Якщо розроблена програмна система виконує різні завдання не на клієнтській машині, а на незалежній обчислювальній платформі, то сторонні постачальники послуг розміщують та управляють послугами програмної системи у хмарі.

Структура надання хмарних сервісів SaaS містить як програмні компоненти (для розробників додатків), так і програми (для користувачів). Шар SaaS часто є сервісно-орієнтованою програмою, тому вона легко інтегрується з іншими додатками SaaS.

Шар надання незалежних обчислень PaaS надає платформу з послугами для розробки, впровадження, тестування, розгортання, моніторингу та хостингу у хмарі. Він не вимагає завантаження або встановлення додаткового програмного забезпечення і підтримує географічно розподілену спільну роботу у високонавантажувальних розподілених системах.

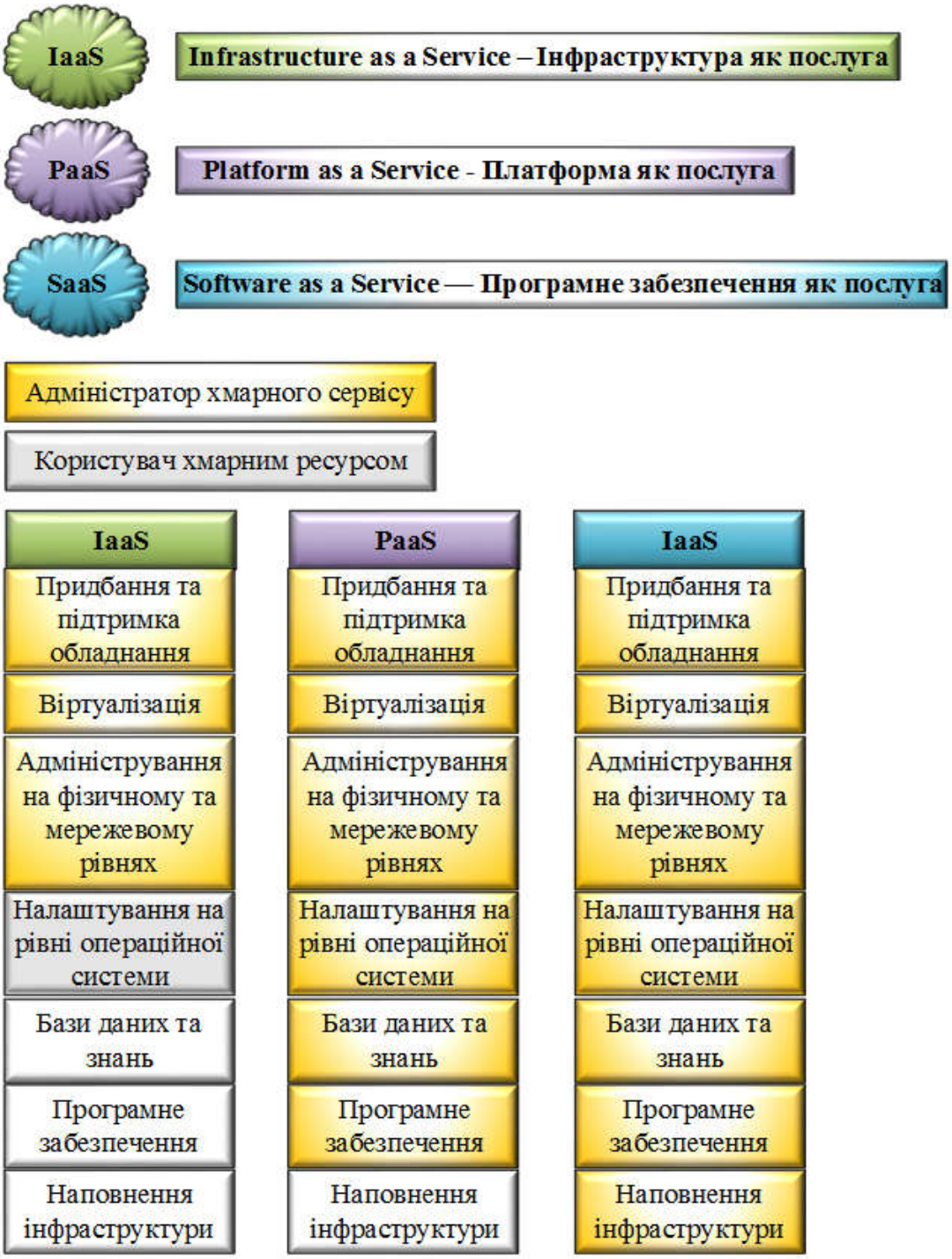


Рисунок 5.3 – Структура надання хмарних послуг

Шар інфраструктури як послуга IaaS віртуалізує обчислювальну потужність незалежних центрів обробки даних, сховища та під'єднання до мережі. Користувачі можуть розширювати ці обчислювальні ресурси на вимогу розробленої програмної системи.

У дисертації використовуються підходи, які ґрунтуються на емпіричних дослідженнях розподілення обчислювального навантаження

через поділ (дроблення) обчислень на більшу кількість модулів. Вихід з робочого стану або значне завантаження одного із обчислювальних модулів на незалежній обчислювальній платформі не є настільки критичним тоді, коли його загальна функціональність мінімізована. Отже, запропонований підхід суттєво збільшує загальний обсяг вузлової та міжмодульної комунікації, тому його використання значно доцільніше для розподілених відмовостійких програмних систем, де втрата інформаційних даних просто неприпустима.

Досліджувані архітектурні підходи та рішення повинні забезпечити мінімізацію втрат надлишкових (незалучених) обчислювальних ресурсів, що є доступними розподіленій відмовостійкій програмній системі на базі хмарних технологій, та автоматизацію рівномірного розподілу обчислювального навантаження між вузлами. Таким чином, ефективний механізм перенесення обчислювального навантаження у програмних системах має забезпечувати мінімізацію втрат неопрацьованих інформаційних даних із хмарних ресурсів під час процедури їх перенесення. Досить важливим критерієм є те, що запропонована стратегія відновлення працездатності програмної системи повинна автоматично визначати причину відмови того чи іншого модуля або вузла загалом.

5.2. Хмарні технології як сервісно-орієнтована архітектура програмних систем

Сервісно-орієнтована інженерія програмних систем містить переваги парадигм послуг, так і хмарних обчислень, пропонуючи користувачам багато переваг для розробки програмних систем, а також і окремих програмних модулів на незалежних обчислювальних платформах. Активне використання цієї технології стрімко актуалізує старі проблеми чинних програмних систем.

Послуги та хмарні обчислення на незалежних обчислювальних платформах привернули увагу як розробників так і працівників, а також і науковців, оскільки ці технологічні рішення дають змогу швидко розвивати широкомасштабні розподілені програмні системні комплекси у таких різноманітних галузях співіснування суспільства, як комплексні дослідження та розробки, електронний бізнес, юстиція, охорона здоров'я, програмні системи з підтримкою сіток, корпоративні обчислювальні інфраструктури, військово промисловий-комплекс

та служба національної безпеки. Використання цих обчислювальних парадигм якісно покращує розробку програмних систем з застосуванням хмарної технології, водночас суттєво мінімізує фінансові витрати на створення та функціонування комерційних програмних систем. Завдяки хмарним технологіям можна вирішувати обчислювальні задачі різного рівня – від простих програмних додатків до комерційних складних критично важливих модулів різних програмних систем [191].

Використовуючи хмарні технології, варто розглянути дві парадигми, які поділяють поняття на такі, як аутсорсинг ресурсів та передача управління програмними системами постачальникам обчислювальних послуг, проте їх особисті акценти при використанні інженерії програмних систем значно відрізняються. Обчислювальні послуги, які надають хмарні ресурси, фокусуються на сервісно-орієнтованій архітектурі під час проєктування, що дає змогу розробляти програмні системи з використанням хмарних платформ за допомогою композиції цих послуг. Хмарні обчислення фокусуються на ефективному використанні та оперативній доставці послуг користувачам за допомогою гнучкої та масштабованої віртуалізації обчислювальних ресурсів і балансування навантаження на незалежних обчислювальних вузлах.

Сервісно-орієнтована інженерія програмних систем (СОІПС) з використанням хмарних технологій містить найкращі з цих двох парадигм. Спочатку СОІПС ґрунтувалася на обчисленнях певних служб, але з часом ця технологія почала активно розвиватися з залученням хмарних обчислень. Сервісно-орієнтована архітектура (SOA), яка є частиною СОІПС, пропонує архітектурний стиль, стандартні протоколи та інтерфейси взаємодії, необхідні для розробки і функціонування програмних систем, а хмарні обчислення забезпечують потрібні пікові потужності при наданні послуг користувачам за допомогою віртуалізації та виділення певного пулу обчислювальних ресурсів на незалежних обчислювальних платформах. Об'єднання послуг та хмарних обчислень у межах інженерії програмних систем може значно допомогти розробникам програмних систем та дати вигідну бізнес-можливість постачальникам таких послуг задовольнити індивідуальні, різноманітні по використанню обчислювальних ресурсів завдання для кожної парадигми. Застосування SOA до етапів життєвого циклу розробки програмного забезпечення, створюючи цикл, що включає не тільки традиційні специфікації вимог, фази проєктування та тестування, а й реалізацію, виявлення та склад послуг. Розробка

програмних додатків для систем в SOA відрізняється від підходів до розробки програмного забезпечення, таких як об'єктно-орієнтоване програмування, розробка програмного забезпечення на основі компонентів та аспектно-орієнтованого програмування.

Конструювання програмних систем з менших компонентів програмного забезпечення в інших методологіях розробки програмного забезпечення статичне, і ручним та залежить від технології й платформи компонентів. На відміну від нього, сервісна композиція в SOA автоматизована за допомогою стандартних протоколів та інтерфейсів, отже, не залежить від конкретної технології або обчислювальної платформи.

Завдання розробки програмних систем із використанням SOA полягає у вирішенні її розподіленого характеру. Не тільки розробляються послуги, що поширюються між різними машинами в різних місцях, а й процес розробки також, оскільки розробники програмних додатків системи, брокери послуг і постачальники послуг працюють незалежно у різних місцях. Отже, ці три зацікавлені сторони повинні співпрацювати через чітко визначені стандарти та інтерфейси на рис. 5.4.

Як і в інших методах розробки програмних систем сервісно-орієнтована інженерія програмних систем починається з програмних вимог. На цьому етапі розробник програмної системи розробляє бізнес-модель, працює з клієнтом для аналізу та уточнення вимог, розробляє робочий процес для бізнес-моделі, а також розкладає вимоги. Розробник високонавантажувальної розподіленої програмної системи потім відправляє кожну розкладену частину вимог до брокера послуг, щоб знайти доступні послуги, які їм відповідають. Після успішного виявлення всіх необхідних служб для задоволення кожної частини програмної системи, розробник програми вибирає необхідні послуги для всіх частин вимог і формує їх у додаток, який створює, робочий процес бізнес-моделі.

Якщо для деяких частин розробленої програмної системи з використанням хмарних технологій не доступні будь-які обчислювальні послуги, тоді розробник програми може зареєструвати їх у каталозі сервісного брокера і почекати, поки потрібні послуги будуть надані. З погляду постачальників послуг, розвиток послуг нагадує інші процеси розробки програмних систем, за винятком того, що послуги також повинні відповідати стандартним протоколам та інтерфейсам взаємодії.

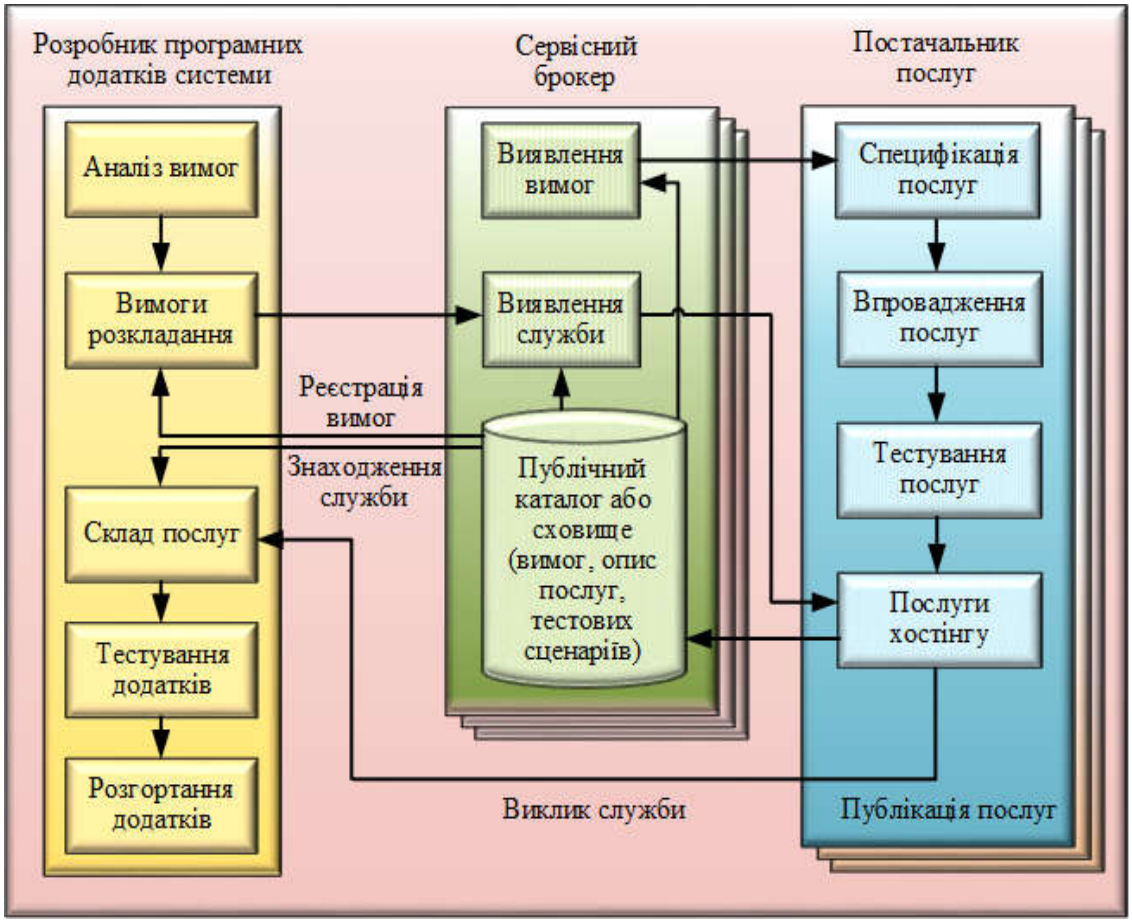


Рисунок 5.4 – Модель взаємодії між розробником додатків, брокерами і постачальниками послуг у сервісно-орієнтованій архітектурі

Розробка програмних систем з використанням хмарної технології на незалежних обчислювальних платформах в SOSE дуже гнучка, оскільки SOA дає змогу публікувати та повторно застосувати не тільки програмні послуги, а й численні артефакти розробки програмних додатків. Розробники окремих програмних додатків системи можуть публікувати бізнес-моделі, шаблони додатків (структури робочих процесів), вимоги, послуги, інтерфейси додатків, засоби тестування, тестування сценаріїв і політики в каталозі брокера, що робить їх доступними для повторного використання. Така гнучкість забезпечує швидкий розвиток широкомасштабних розподілених програмних систем з використанням хмарної технології.

Хоча СОІПС є концептуально новим та перспективним напрямом, її реалізація потребуватиме додаткових досліджень у галузі розробки програмних систем для вирішення нагальних проблем, таких як управління безпекою та якістю обслуговування (QoS) користувачів, що виникають у сфері послуг або хмарних обчислень на незалежних платформах.

Розробники сервісів при розробці програмних систем дотримуються сервісно-орієнтованої архітектурної моделі для створення та оптимального обміну обчислювальними процесами і ресурсами, упакованими як послуги. Кожна послуга є незалежним програмним об'єктом конкретної системи з чітко визначеним стандартним інтерфейсом, який забезпечує певні функції при обміні даними у розподілених високонавантажувальних системах з використанням хмарної технології. Розробники таких програмних систем можуть динамічно створювати послуги, як робочий процес, який є основою програм та обчислювальної системи у загалом. У цьому контексті сама програмна система з використанням хмарних технологій може бути користувацькою послугою – автономним, бездержавним та незалежним від платформи об'єктом з URL, інтерфейсом і функціями, які можуть бути описані і наведені як XML – стандартизовані дані.

Використання хмарних технологій при розробці програмних систем має достатньо різноманітний спектр організації з різною політикою розробки і методів використання та керування послугами. Угоди на рівні надання обчислювальних послуг визначають вимоги до середовища виконання, які регулюють взаємодію служби з користувачем – одержувачем послуг – або з іншими службами. Служба SLA описує особливості цієї послуги і визначає терміни її дії, по суті перетворюючись на договір про надання користувацьких послуг на незалежній обчислювальній платформі, які постачальники (хмарних технологій) зокрема повинні виконувати.

Використовуючи стандартні протоколи та інтерфейси взаємодії користувача при використанні хмарних незалежних обчислювальних платформ, розробники програмних систем можуть динамічно реагувати на такі дії, як: виявляти, створювати, перевіряти, та виконувати служби у своїх додатках програмних систем під час їхнього виконання. Розробка програмних систем на основі сервіс-орієнтованої архітектури здійснюється завдяки виявленню та коректній композиції послуг, яка включає три зацікавлені сторони:

1. Постачальник послуг (або розробник програмних систем) є стороною, яка розробляє і приймає послугу.
2. Споживач послуг – це людиною або програмна система, яка використовує службу для створення програми.
3. Брокерське обслуговування допомагає постачальникам публікувати та продавати свої послуги і допомагає споживачам знаходити та використовувати доступні ресурси.

Розробникам високонавантажувальних розподілених програмних систем не потрібно інтегрувати службовий код у додатки, тому що служба працює на сайті свого постачальника і вільно інтегрується з додатками через стандартні протоколи обміну повідомленнями. Отже, послуги та програми не повинні використовувати одну мову програмування або працювати на одній платформі. На відміну від програмної системи з використанням хмарних технологій на незалежних обчислювальних платформах, яка надає користувальницький інтерфейс, служба зазвичай надає інтерфейс прикладного програмування (API), щоб програма системи або інші служби могли викликати цю службу. Як випливає з вище наведеного детального опису, послуги мають кілька привабливих характеристик, такі як:

- слабо пов'язані між собою – не існує прямих залежностей між окремими службами;
- абстрактні – поза описом SLA послуга приховує свою бізнес-логіку від зовнішнього світу;
- багаторазові послуги – спрямовані на багаторазову підтримку потенційного повторного використання обчислювальних ресурсів;
- композитна – послуга може містити комплекс інших, а розробники програмних систем можуть координувати та збирати для формування композиту та оперативного вирішення нагальних потреб;
- без застави – щоб залишатися вільно пов'язаними, послуги не підтримують інформацію про стан та специфічну її діяльність, наприклад запит на обслуговування;
- відомий – сервіс, нехай послуга є механізмом для використання споживача, щоб виявити і зрозуміти їх опис.

Загалом набір цих характеристик дає змогу активно розробляти високо навантажувальні розподілені програмні системи з використанням хмарних технологій для обчислення користувацьких потреб. Всесвітні органи з питань стандартизації, такі як Організація з удосконалення структурованих інформаційних стандартів (OASIS) та Консорціум Всесвітньої мережі (W3C), створили різноманітні протоколи обміну даними та сервісні інтерфейси, що дають змогу розробляти програмні системи з використанням незалежних обчислювальних платформ за допомогою сервіс-орієнтованої архітектури. У табл. 5.1. наведено основні протоколи та інтерфейси високонавантажувальних розподілених програмних систем.

Таблиця 5.1 – Основні протоколи та інтерфейси сервісно-орієнтованої інженерії програмних систем

Protocols and interfaces thet enable SOSE		
Protocol or interface	**Standards body (if applicable)**	**Purpose**
XML	W3C	Represent data in SOA
Simple Object Access Protocol (SOAP)	OASIS	Invoke servises remotely across networks and platforms
Representational State Transfer (REST)	Architectural stule, not a standard (attributed to Roy Fielding)	Invoke servises remotely across networks and platforms
Web Services Description Language (WSDL)	W3C	Describe interfaces and servise functional
Universal Description, descovery, and integration (UDDI)	OASIS	Automatically publish and discover services
Electronic business XML (ebXML)	OASIS and United Nations Center for Trade Facilitation and Electronic Business (UN/CEFACT)	Automatically publish and discover
Bisiness Process Execution Language (BPEL)	OASIS	Orchestrate services in a workflow

Хмарні технології з використанням незалежних обчислювальних платформ забезпечують зручний та якісний мережний доступ за користувацьким запитом до загального незалежного пулу конфігурованих обчислювальних ресурсів, таких як мережі, сервери, сховища, програми та служби, які хмарна програмна система може швидко надавати за будь-якою вимогою у будь-коли та звільняти надані ресурси автоматично, якщо вони більше непотрібні. Хмарні обчислення дають змогу споживачеві (користувачеві або програмній системі) залучати обчислювальні можливості за потреби у мережах будь-коли і будь-де.

5.3. Хмарні сервіси на незалежних обчислювальних платформах

Зі стрімким розвитком інформаційних технологій набули подальшого розвитку й інші моделі надання та використання хмарних послуг на незалежних обчислювальних платформах(рис. 5.5).

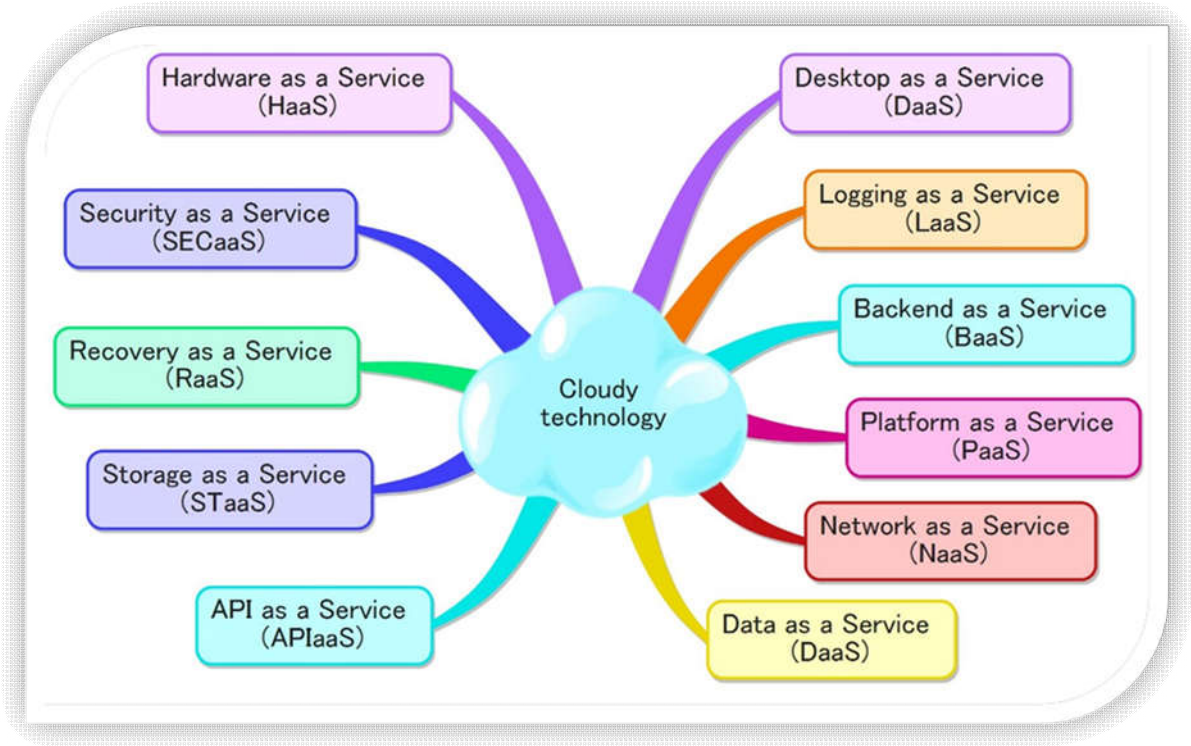


Рисунок 5.4 – Модель хмарних сервісів на незалежних обчислювальних платформах

Наслідок еволюційного розвитку наявних моделей послуг зокрема:

- Hardware as a Service (HaaS) – апаратне забезпечення;
- Security as a Service (SECaaS) – надання безпеки;
- Backend as a Service (BaaS) – «бекенд»;
- Recovery as a Service (RaaS) – відновлення програм та даних;
- Data as a Service (DaaS) – отримання та опрацювання даних;
- Logging as a Service (LaaS) – авторизація та ідентифікація;
- Network as a Service (NaaS) – мережеві технології;
- Platform as a Service (PaaS) – обчислювальна платформа;
- Desktop as a Service (DaaS) – робочий стіл;
- Storage as a Service (STaaS) – сховища та банки даних;
- API as a Service (APIaaS) – API (прикладний програмний інтерфейс).

Деякі з перелічених сервісних послуг, які надаються незалежними обчислювальними платформами спрямовані на використання обмеженим колом спеціалістів таких як, розробники й адміністратори, інші ж успішно використовують споживачами у широкому спектрі своїй діяльності.

Розробка програмних систем з використанням хмарних технологій стосується не лише програмних модулів, а й забезпечення їхнього ефективної доставки розробленого програмного модуля, як складової програмної системи зацікавленим користувачам, що у водночас містить етапи розгортання та обслуговування програмних модулів. Проте сервісно-орієнтована архітектура не розглядає, як розроблений додаток має бути доставлений зацікавленим користувачам, або як незалежні постачальники обчислювальних послуг ефективно будуть керувати програмними модулями під час виконання. Хмарні обчислення на незалежних платформах можуть якісно посприяти SOSE виконати ефективну доставку програмних модулів шляхом забезпечення зручного їхнього розгортання й обслуговування постачальниками хмарних послуг через віртуалізацію останніх, використовуючи стандартизований інтерфейс для зручного доступу зацікавлених користувачів та використання програмних модулів як частини розробленої програмної системи з використанням хмарних технологій.

Виклики для розробки програмних систем

Сучасні парадигми у розробці програмних систем з використанням незалежних обчислювальних платформ вимагають інноваційних підходів до забезпечення ефективної віртуалізації та взаємодії між шарами SaaS, PaaS та IaaS. Суть такого підходу вимагає перегляду нагальних питань з інженерії розробки програмних систем, проте деякі з яких не є новими, водночас вони вимагають більш серйозної уваги у контексті надання обчислювальних послуг на хмарних платформах. Ми відзначили сім критичних областей при розробці програмних систем, які створюють серйозні перешкоди для розробки програмних модулів за допомогою SOSE.

Конфіденційність і цілісність даних

Використовуючи незалежні обчислювальні платформи у хмарних обчисленнях, зацікавлені користувачі отримують обмежений контроль над опрацюванням та зберіганням інформаційних потоків даних, тобто на віддалених обчислювальних вузлах, якими володіють та на яких працюють різні постачальники цих послуг. Оскільки ці інформаційні потоки даних не зашифровані, то існує значний ризик, що постачальники хмарних послуг або шкідливі користувачеві програми можуть розкрити або змінити їх вміст. Попри те, що існує багато методів по захисту конфіденційності, усі вони не можуть повністю гарантувати його, проте застосовуючи певну модель захисту до служб і

систем з надання хмарних обчислень потрібно пам'ятати, що вони лише призначені для захисту інформаційних потоків даних від шкідливих атак поза системою. Служби та програмні системи хмарних обчислень на незалежних платформах мають різноманітних постачальників цих послуг всередині віртуальної систем.

Надійність та доступність хмарних платформ як послуг

Для вирішення бізнес-логіки підприємств зацікавлені особи значною мірою покладаються на незалежних постачальників хмарних послуг. У інформаційному просторі з'являються серйозні занепокоєння щодо того, як ці інформаційні загрози можуть вплинути на надійність та доступність хмарних платформ як послуги – від нестабільного економічного стану суспільства у цілому та постачальника послуг зокрема до природних та техногенних катастроф й кібератак – можуть суттєво впливати на служби, а отже, і на бізнес користувача хмарних послуг.

Щоб мінімізувати ці інформаційні загрози, користувачі хмарного сервісу незалежного постачальника повинні перевірити свій план резервного копіювання інформаційних потоків даних, надійність системи загалом, плани виходу із надзвичайних ситуацій та відновлення програмної системи до працездатного стану, підтримку при завершенні служби та документування історії подій, перш ніж приймати рішення про використання певних послуг.

Сьогодні надзвичайно критичною загрозою є різноманітні кібератаки. Послуги та системи хмарних обчислень незалежних постачальників оперативно і гнучко забезпечують масиви обчислювальних ресурсів відповідно до їхніх бізнес-вимог зацікавлених користувачів. Для бізнес-спільноти обчислювальні можливості та ресурси незалежних постачальників хмарних послуг часто здаються необмеженими, оскільки вони доступні для використання у будь-коли і в будь-якому обсязі. Однак, програмні засоби інформаційних кібератак також можуть купувати значні обсяги хмарних обчислювальних ресурсів, які дають їм змогу запустити більш потужні кібератаки. Зловмисники вже використовували хмарні обчислювальні платформи Amazon EC2 і Google AppEngine.

Для розв'язання цієї проблеми як для самих послуг, так і для незалежних постачальників хмарних сервісів, потрібні інноваційні та ефективні технічні засоби програмного забезпечення для моніторингу

і виявлення шкідливих дій щодо користувачів, а також для суворої автентифікації користувачів й контролю їх доступу.

Безпека у хмарному середовищі багатозадачності

У розроблених програмних системах з використанням хмарних технологій на незалежних обчислювальних платформах у мультифункціональному пристрої один екземпляр програмного модуля працює на сервері, який одночасно може мати декілька користувачів або орендарів. У багатошаровій архітектурі розроблена програмна система практично розділяє свої інформаційні дані та конфігурацію, де кожен зацікавлений користувач працює з індивідуальним зразком віртуального програмного модуля.

Служби та системи хмарних обчислень на незалежних обчислювальних платформах мають мультиоренду, оскільки чимало зацікавлених користувачів багаторазово використовують програмний модуль та набір апаратних засобів для опрацювання інформаційних потоків даних. Головною проблемою застосування хмарних платформ із захисту інформаційних потоків даних та розроблених програмних систем є вразливість до кібератак. Для забезпечення необхідного рівня захисту постачальники послуг використовують гіпервізори, які контролюють доступ між віртуальними машинами та апаратно-програмними засобами. Проте деякі апаратно-програмні засоби, такі як кеші процесорів і графічні процесори, не призначені для забезпечення сильних ізоляційних властивостей для багатошарової архітектури. Навіть гіпервізори віртуальних машин, які надаються незалежним постачальником хмарних послуг, можуть мати певні недоліки, які дають змогу віртуальній машині одного зацікавленого користувача отримувати несанкціонований контроль над іншими задачами та потоками даних. Ще не так давно зловмисники використовували численні вразливості гіпервізорів для впливу на обчислювальні операції інших користувачів або для отримання несанкціонованого доступу до масивів даних. Отже, вирішення цих вразливостей вимагає інноваційних методів розробки програмних систем для забезпечення багатошарових архітектур при використанні служб на хмарних платформах, зокрема ізоляції та моніторингу віртуальних машин.

Профіль ризику при використанні хмарних технологій

У розроблених програмних системах з використанням хмарної технології на незалежних обчислювальних платформах користувачі мають обмежений доступ до послуг та хмарних обчислень, а також до

інформації про архітектуру внутрішньої системи, версії програмних модулів, конфігурації, операції та відповідні практики безпеки постачальників цих послуг. Такий обмежений доступ зацікавлених користувачів може підвищити зручність використання, але він також має серйозні недоліки для управління операційними ризиками. Управління ризиками в інженерії програмних систем гарантує, що розробники програмних модулів на початковому етапі виявляють та аналізують загрози для бізнес-процесу розробки програмних модулів, а також використовують відповідні стратегії для мінімізації й контролю ризиків. А саме, як невдача завершення проєктів у межах зазначених часових графіків та бюджетних обмежень, а також не у повному об’ємі задоволення вимог зацікавлених користувачів. Оскільки розробникам програмних систем з використанням хмарної технології не вистачає інформації про внутрішню організацію системи під шаром віртуальної абстракції, вони можуть бути позбавлені можливості провести відповідне дослідження по управлінню ризиками. Щоб вирішити цю проблему, розробники програмних систем повинні звернутися до незалежних постачальників хмарних послуг із розглядом трьох етапів:

- часткове або повне розкриття інформації про дизайн програмного забезпечення та інфраструктуру;
- розкриття відповідних журналів та даних, таких як журнали вторгнення в мережу, журнали виявлення аномалій і журнали подій безпеки;
- розкриття деталей політики безпеки та механізмів правозастосування.

Фаховий розгляд цих етапів не усуне повністю ризик, проте отримана інформація забезпечить значно ефективніше управління бізнес-ризиками.

Моніторинг якості надання хмарних послуг

Незалежні постачальники хмарних платформ як послуг та управління різноманітними програмними вимогами QoS надзвичайно важко керувати, оскільки численні розробники програмних модулів динамічно створюють послуги у мережах для формування декількох обчислювальних робочих процесів, а різні постачальники хмарних технологій з різноманітними методами та політиками по різному керують службами. Отже, функції QoS всіх хмарних служб тісно пов'язані, і між ними є компроміси.

Отже, функції, які відповідають за пропускну здатність та затримку певної служби, покладаються на розподіл інформаційних ресурсів розробленої програмної системи під час виконання модуля. Часто на одному сервері розміщено кілька служб, які конкурують між собою за процесорним часом, пам'яттю та пропускною здатністю сервера. Крім того, композиції сервісів, стан ресурсів сервера, пріоритети робочих процесів і вимоги QoS зазвичай динамічно змінюються під час виконання хмарних обчислень. Отже, задоволення вимог QoS для декількох обчислювальних процесів вимагає ефективних методів адаптивного розподілу системних ресурсів для кожної хмарної служби. Для управління кількома властивостями QoS для таких розроблених програмних систем, сервісів і систем хмарних обчислень потрібна ситуаційна обізнаність, аналіз контексту й оцінка QoS, а також оптимальний розподіл програмно-апаратних ресурсів.

Інтернет-речі як системи отримання хмарних послуг

При розробці програмних систем з використанням хмарної технології основним критерієм є мережі, оскільки користувачі або програмні системи розміщені на різних апаратних пристроях, а саме: настільних комп'ютерах, ноутбуках, смартфонах, планшетах та персональних комп’ютерах, тобто можуть отримати доступ до мережевих послуг у будь-коли та будь-де за допомогою стандартних протоколів обміну інформаційними даними. Оскільки крадіжки особистих даних та викрадення послуг є основними загрозами, то мобільні послуги та постачальники послуг з обчислювальної техніки у незалежних хмарних платформах потребують жорстких методів розробки програмних систем для забезпечення не обмеженого доступу до обчислювальних послуг та даних.

Законодавча база використання хмарних платформ

Користувачі, які застосовують послуги та системи хмарних обчислень на незалежних платформах, не знають точного фізичного та географічного розташування своїх інформаційних даних, оскільки опрацювання та зберігання даних часто перебувають у невизначених місцях, як внутрішніх, так і зарубіжних. Проте юридично кожна територія має іншу законодавчу юрисдикцію, а незалежні постачальники хмарних послуг у зарубіжних країнах не завжди можуть гарантувати відповідність нормативним та правовим вимогам. Це – може бути, як захист у конфіденційності, резервному копіюванню інформаційних даних або наданню аудиторської перевірки тощо. Отже, незалежні

постачальники хмарних послуг можуть бути не готові взяти на себе відповідальність за інциденти безпеки, за невиконання вимог резервного копіювання даних, за надання аудиторських перевірок. Вони також можуть і не захищати інтелектуальну власність згідно зі стандартами відповідності. Розробники програмних систем, які встановлюють угоди SLA з провайдерами хмарних послуг, повинні перевірити, чи постачальники зобов'язуються зберігати та опрацьовувати інформаційні дані у певних територіальних юрисдикціях. Незалежні постачальники хмарних послуг повинні зробити контрактне зобов'язання із дотриманням всіх регуляторних вимог і зобов'язань при публікації та управління програмними системами з використанням хмарної технології.

Незважаючи на те, що обчислювальні послуги та хмарні обчислення мають великі перспективи, проте виникають і деякі критичні зауваження щодо дотримання дедалі більших вимог до динамічної розробки та використання програмних модулів, а повна реалізація цього потенціалу вимагає зміни структури розробки програмних систем. Програмна інженерія систем може допомогти об'єднати ці обчислювальні парадигми та використати їх значні переваги для розробки програмних модулів. Хоча багато проблем залишається у перенесенні цієї ідеї до технічної реалізації, переваги такого середовища повинні слугувати мотивацією для досліджень з розробки програмних систем з використанням хмарних технологій, які можуть вирішити ці нагальні інформаційно-комунікативні проблеми.

5.4. Проєктний менеджмент як хмарний сервіс у розробленні програмних систем

У розробленні програмних систем з використанням хмарної технології є важливим фактором проєктний менеджмент, який забезпечує взаємодію і групову роботу над поставленим завданням. Для забезпечення успішного управління розробленням програмної системи, необхідно розгорнути у незалежного постачальника хмарних послуг систему управління проектним менеджментом. Розгортання цієї програми як сервісу є одним із важливих етапів життєвого циклу будь-якого хмарного модуля. На цьому етапі необхідно приділити основну увагу розробленню та впровадженню відповідної архітектури розгортання, яка дала б змогу забезпечити безперервну роботу усіх його

програмних модулів. Основним критерієм стабільної роботи програмної системи з використанням хмарної технології є надійність та масштабованість як основні вимоги до якісної програмно-апаратної архітектури розгортання хмарних сервісів. Наш хмарний сервіс складається з таких компонентів: Web-програмний модуль; база та банк даних; двигун повнотекстового пошуку інформаційних потоків даних; сховище контенту.

Для розгортання незалежного хмарного сервісу для проєктного менеджменту використаємо Redmine, а для його коректного функціонування потрібно забезпечити відповідне оточення на віддаленому сервері. Redmine є відкритим серверним Web-додатком для управління проєктними програмними розробками та системами, а також використовується для моніторингу і відстеження помилок. Прикладний програмний модуль Redmine, який написаний мовою програмування Ruby, і є програмним компонентом на основі відомого веб-фреймворку Ruby on Rails.

Функціональні можливості прикладного додатку Redmine:

ведення декількох незалежних програмних проєктів із розробки програмних систем;

гнучка система колективного доступу, заснована на ролях;

система фіксації та відстеження помилок;

ведення діаграм Ґанта та щогодинний календар виконання робіт;

відстеження планових та нових завдань щодо програмного проєкту, а також документів і управління файлами;

миттєве сповіщення всіх зацікавлених осіб про корегування програмного проєкту за допомогою RSS-потоків та електронної пошти;

організація кооперативних форумів для кожного програмного проєкту;

ведення та фіксація обліку тимчасових витрат із розробки прикладного програмного додатку;

можливість додаткового налаштування довільних полів для інцидентів, а також тимчасових витрат на проєкт і користувачів;

забезпечення легкої інтеграції із системами управління різними версіями (Mercurial, SVN, Bazaar, CVS, Git, і Darcs);

створення журналу записів про помилки та відхилення від проєктних планів на основі отриманих повідомлень;

забезпечення множинної підтримки автентифікації LDAP;

можливість самостійного корегування персональних даних користувачів та реєстрації нових;

підтримка багатомовного інтерфейсу;

забезпечення інтеграції систем керування базами даних Microsoft SQL Server, MySQL, PostgreSQL, SQLite, Oracle.

Групову роботу з розробки програмних систем з використанням хмарної технології та управління під час роботи над програмними проєктами можна розглядати як систематичний процес створення, зберігання та розподілу людських та технічних ресурсів. У дослідженні розглядаються актуальні питання з управління розробкою програмних систем у розрізі глобальної програмної інженерії та особливості командної співпраці (групової роботи) всіх учасників проєкту 300, 310, 311, 323].

Методології менеджменту при розробці програмних систем сьогодні трансформуються у певні міжнародні стандарти:

Стандарт з організації компетенцій;

Стандарт з оцінки ефективної діяльності у проєктах;

Стандарт за індивідуальними компетенціями для управління проєктами, програмами та портфелями проєктів.

Сьогодні немає ефективної структури знань про методологію розробки управління проєктами програмних систем проте вдосконалюються міжнародні стандарти «P16326:201x WD5, Dec 2017 – ISO/IEC/IEEE International Standard – Systems and Software Engineering – Life Cycle Processes – Project Management» IEEE 12207 та ІСО/МЕК, IEEE 15288:2015. Цей стандарт використовують менеджери програмних проєктів з надання фахової підтримки управління та успішного завершення прикладних програмних проєктів [328]. Міжнародний стандарт розширює ґрунтовне керівництво проєктом для успішного управління і маніпулювання технічними процесами під час розробки прикладних програмних систем. Використання програмного модуля Redmine в середовищі ОС Windows як безкоштовного хмарного сервісу, який забезпечує груповий менеджмент управління та швидку реакцію на будь-які зміни як внутрішнього, так і зовнішнього оточення під час розробки прикладної програмної системи, стає критичним чинником успіху всього проєкту. Структура бази даних сервера Redmine наведена в табл. 5.2

Таблиця 5.2 – Структура бази даних сервера Redmine

Принципи	Коротка характеристика принципів
1	2
Користувачі системи	Користувачі сервера Redmine є одним із основних понять предметної області. Модель функціонування є основою для ідентифікації та автентифікації користувача, що працює з системою персоналу із членів команд розробників, а також для розподілення їх навантаження у різних ролях проєкту
Ролі	Ролі членів команд визначаються адаптивною моделлю призначення прав доступу до системи адміністрування менеджментом розробки програмної системи. Ролі містять набори прав та привілеїв, що дають змогу розмежовувати доступи до різноманітних функціональностей системи. Членам команди призначають відповідну роль у кожному проєкті окремо, в якому він бере участь. Член команди розробників програмних систем може мати декілька ролей, а призначення ролі для окремого завдання (issue) зараз неможливо
Проєкти	Проєкт є одним з основних понять у цій предметній сфері по розробці програмних систем з використанням хмарної технології, а саме – систем оперативного управління проєктними розробками. Завдяки цьому поняттю можна налагодити командну роботу і планування людських та фінансових ресурсів у декількох проєктних розробках одночасно з розмежуванням рівня прав доступу різним членам команд. Управління проєктами допускає ієрархічну вкладеність завдань та задач
Трекери	Трекери – базова класифікація, за якою впорядковуються завдання у структурі програмного проєкту. Поняття «трекер» входить до систем обліку помилок (англ. Bug tracking tool), в якому відображено структуру кожного окремого проєкту. Отже, в Redmine трекери – аналог підкласів класу «Завдання» та є основою для поліморфізму різних завдань, а також дають змогу визначати різні типи полів. Прикладами трекерів є: «Помилка», «Поліпшення», «Підтримка», «Документування»

Продовження табл. 5.2

1	2
Завдання	Завдання є базовим поняттям всієї системи управління проєктами, що описують поставлену задачу, яку необхідно виконати, та хто її буде виконувати. У кожному завданні (task) є такі атрибути, як опис, автор, та обов'язково воно має бути прив'язане до трекера. Кожне завдання набуває певного статусу. Статус – це окрема сутність із можливістю визначення прав та привілеїв для різних ролей певного члена команди розробників. Для кожного програмного проєкту необхідно окремо визначити набір етапів (послідовності) розробки та набір категорій завдань
Відстеження зміни статусу завдань	За контролем та корегуванням параметрів завдань проєкту членами команди розробників у системі сервера Redmine відповідають дві сутності: «Змінений параметр» та «Запис журналу змін». Сутність «Змінений параметр» відображається в окремому журналі та призначена для фіксації старого і набутого значення членом команди розробників параметра. Запис у журналі відображає лише одну дію члена розробників щодо корегування параметрів завдання або додавання коментаря до нього. Отже, запис може слугувати одночасно інструментом як ведення історії завдання проєкту, так і ведення діалогу між членами команди розробників програмних систем з використанням хмарної технології
Зв'язки між завданнями	Задачі всередині проєкту можуть бути пов'язані між собою, наприклад, одна задача може мати декілька вкладених підзадач або передувати іншій. Ця інформація використовується у процесі планування розробки прикладних програмних додатків, а за її збереження в Redmine відповідає окрема сутність

Продовження табл. 5.2

1	2
Облік витраченого часу на проєкт	У Redmine є система моніторингу та обліку витраченого часу, що контролює «Витрачений час», який пов'язаний із членами команд та часом, витраченим на ту чи іншу задачу. Ця сутність дає змогу контролювати витрачений час та діяльність члена команди розробників (розробляння, проєктування, підтримка) і додавати короткі коментарі до актуальних задач. Цю інформацію можна використати для контролю витраченого часу та оплати праці членів команд розробників програмних систем
Прив'язка репозиторіїв	Сервер Redmine дає змогу ефективно та оперативно інтегрувати з різноманітними прикладними програмами контролю версій (репозиторіями). Інтеграція та моніторинг дають змогу відстежувати зміни у зовнішньому репозиторії та фіксацію цих змін у базі даних, проводити аналіз змін у проектній розробці з метою контролю їхнього та прив'язки до завдань. В інфологічній моделі структури бази даних системи менеджменту проєктів за кооперацію із зовнішніми репозиторіями відповідають три сутності, а саме: «Сховище», «Редакція» і «Зміна»
Отримання повідомлень	Отримання повідомлень користувачем про зміни, які відбулися у проєктній розробці, відбуваються у сутності «Спостерігачі», що позв'язує членів команд розробників з об'єктами різноманітних класів (проєкти, задача, завдання, форуми тощо)

5.5. Розгортання хмарного сервісу для управління розробленням програмної системи

Розгортання хмарного сервісу на незалежних обчислювальних платформах такого сервера, як Redmine, для управління проєктами розробки програмних систем. Для завантаження сервісу управління розробками Redmine на незалежну хмарну обчислювальну платформу необхідно встановити програмне середовище Ruby. Алгоритм моні-

торингу інсталяції встановлення Ruby відображено на рис. 5.6. Встановлюючи середовище, активізуємо прапорці на «Add Ruby executables to your PATH» та «Associate .rb and .rbw files with this Ruby installation».

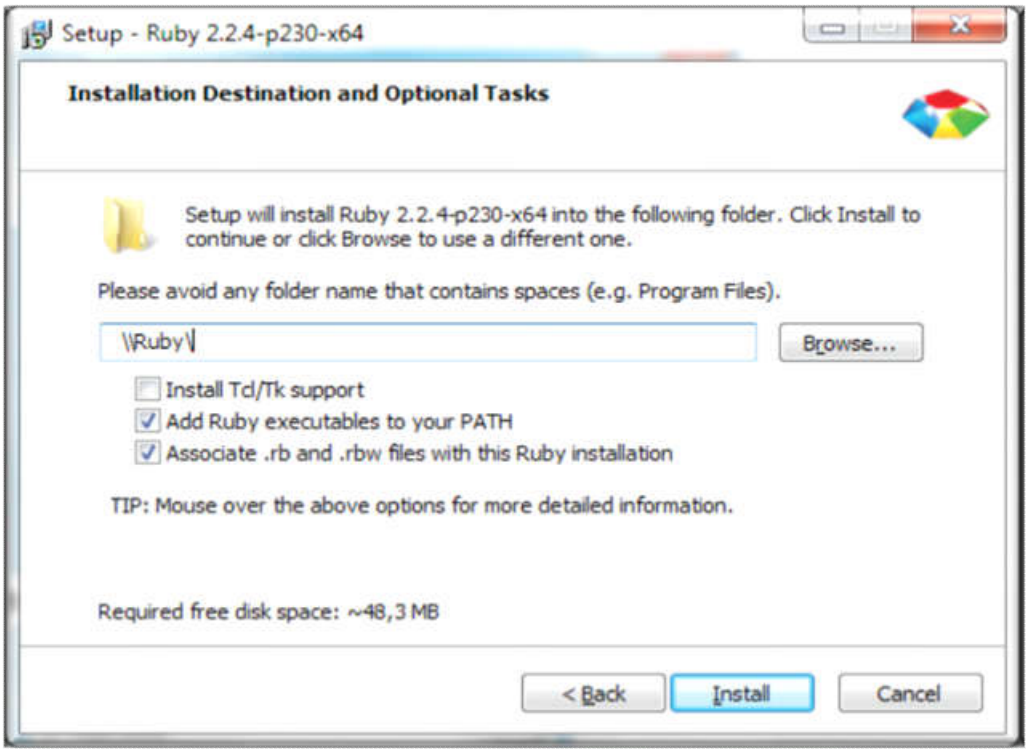
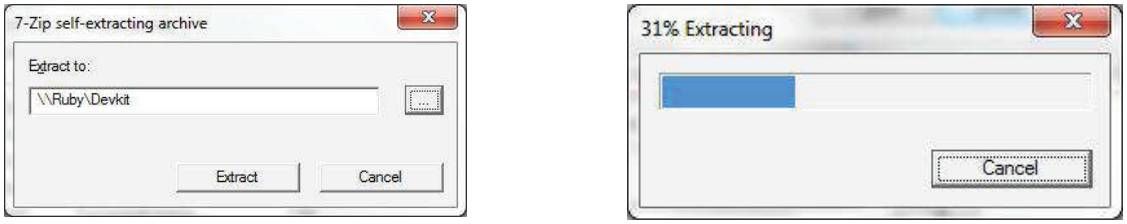


Рисунок 5.6 – Моніторинг встановлення Ruby

Також для розгортання хмарного сервера Redmine як сервісу необхідно встановити Ruby DevKit 4.7.2. У консолі переходимо у папку з DevKit і виконуємо команду ruby dk.rb init і ruby dk.rb install (рис. 5.7). Завантажуємо Redmine як хмарний сервіс на незалежну обчислювальну платформу та розгортаємо його у папці, де встановлений Ruby, а також копіюємо сертифікат безпеки з Web-сторінки у робочий файл \\Ruby\lib\ruby\2.1.0\rubygems\ssl_certs.

Наступним етапом розгортання хмарного сервісу на сервері Redmine є встановлення менеджера керування «ґемами» за допомогою команд «gem install bundler» та SQLite для мови програмування Ruby «gem install sqlite3-ruby». Встановлюємо полегшену версію керування реляційними базами даних на основі мови запитів SQL, далі оновлюємо всі «ґеми» за допомогою команд «gem update» (рис. 5.8).



а) б)

```
\\Ruby\DevKit>ruby dk.rb init
Initialization complete! Please review and modify the auto-generated
' config.yml ' file to ensure it contains the root directories to all of the
installed Rubies you want enhanced by the DevKit.
\\Ruby\DevKit>ruby dk.rb install
[ INFO ] Skipping existing gen override for ' \\Ruby '
[ WARN ] Skipping existing DevKit helper library for ' \\Ruby '
```

в)

Рисунок 5.7 – Моніторинг встановлення Ruby DevKit

```
\\Ruby\DevKit>gem install bundler
Fetching: bundler-1.11.2.gem <100%>
Successfully installed bundler-1.11.2
Parsing documentation for bundler-1.11.2
Installing ri documentation for bundler-1.11.2
Done installing documentation for bundler after 16 seconds
1 gem installed
```

а)

```
\\Ruby\DevKit>gem install sqlite3-ruby
Fetching: sqlite3-1.3.11-x64-mingw32.gem <100%>
Successfully installed sqlite3-1.3.11-x64-mingw32.gem
Fetching: sqlite3-ruby-1.3.11.gem <100%>
```

б)

```
\\Ruby\DevKit>gem update
Updating installed gems
```

в)

Рисунок 5.8 – Оновлення «gem» командою «gem update»

Для налаштування конфігурації хмарного сервісу Redmine потрібно провести його конфігурування, тобто: перейти у каталог config, відкрити файл database. yml.example для редагування, видалити усі записи, крім тих, які стосуються SQL3, і зберігти під новою назвою database.yml. У кореневому каталозі Redmine відкриваємо конфігураційний файл GemFile, у якому на початку прописуємо «gem «sqlite3-ruby»». Наступним кроком налаштування хмарного сервера Redmine є перехід у кореневу папку та виконання команди «bundle install» (рис. 5.9).

```
\\Ruby\redmine>bundler install
Using rake 11.0.1
Using i18n 06.11
Using multi_json 1.11.2
Using builder 3.0.4
Using erubis 2.7.0
Using journey 1.0.4
Using rack 1.4.7
Using hike 1.2.3
Using tilt 1.4.1
Using mine-types 1.25.1
Using polyglot 0.3.5
Using arel 3.0.3
Using tzinfo 0.3.46
Using coderay 1.1.1
Using htmlentities 4.3.1
Using json 1.8.3
Using thor 0.19.1
Using net-ldap 0.3.1
Using rails 3.2.22
Bundle complete! 28 Gemfile dependencies, 41 gems now installed.
Gems in the groups development, test and rmagick ware not installed.
Use 'bundle show [gemname]' to see where a bundled gem is
installed.566666666xxx
```

Рисунок 5.9 – Консольне вікно інсталяції «bundler»

Для подальшого налаштування виконуємо команду «rake generate_secret_token» (рис. 5.10).

```
\\Ruby\redmine>rake generate_secret_token
\\Ruby/lib/ruby/gems/2.2.0/gems/htmlentities–
4.3.1/lib/htmlentities/mappings/expanded.rb:465 warning: duplicated
key at line 466 ignored: "inodot"
```

Рисунок 5.10 – Консольне вікно виконання команди «rake generate_secret_token»

Також для встановлення хмарного сервісу на сервері Redmine на незалежній обчислювальній платформі необхідно виконати команди «rake redmine: load_default_data RAILS_ENV=«production» та команду «ruby script/rails s – e «production» (рис. 5.11).

Внаслідок проведених операцій встановлення сервера Redmine як хмарного сервісу на незалежних обчислювальних платформах для проєктного управління та моніторингу розробки програмних систем з використанням хмарної технології отримаємо безкоштовний Web-сервіс. Результат обчислень операцій з розгортання сервера Redmine як незалежного хмарного сервісу зображено на рис. 5.12.

```
\\Ruby\redmine>rake redmine: load_default_data RAILS_ENV=
production
\\Ruby/lib/ruby/gems/2.2.0/gems/htmlentities–
4.3.1/lib/htmlentities/mappings/expanded.rb:465 warning: duplicated key
at line 466 ignored: "inodot"
Select language: ar, az, bg. bs. ca. cs. da. de. el. en. en–GB, es, et. Eu.
Fa, fi, fr, gl, he, hr, hu, id, it, ja, ko, lt, lv, mk, mn, nl, no, pl, pt, pt–BR,
ro, ru, sk, sl, sq, sr, sr–YU, sv, th, tr, uk, vi, zh, zh–TW, [tn] uk
```

а)

```
\\Ruby\redmine>ruby script/rails s –e "production"
\\Ruby/lib/ruby/gems/2.2.0/gems/htmlentities–
4.3.1/lib/htmlentities/mappings/expanded.rb:465 warning: duplicated key
at line 466 ignored: "inodot"
 => Booting WEBrick
 => Rails 3.2.22 application starting in production on http://0.0.0.0:3000
 => Call with –d to detach
 => Ctrl-C to shutdown server
```

б)

Рисунок 5.11 – Консольне вікно виконання команди «ruby script/rails s – e «production»

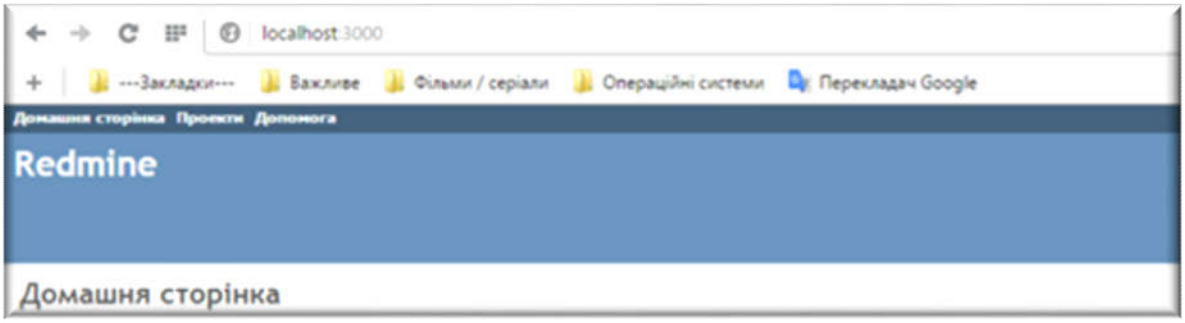


Рисунок 5.12 – Робоче вікно сервера Redmine як хмарного сервісу

Для професійного використання сервера Redmine як хмарного сервісу на незалежних обчислювальних платформах необхідно встановити та налаштувати плагін Backlogs (рис. 5.13):

1. Розпаковуємо папку з плагіном «\\ruby/redmine/plugins/».
2. Запускаємо на виконання команду «set RAILS_ENV=production».
3. Виконуємо інсталяцію плагіну командою «bundle exec rake redmine:backlogs: install».

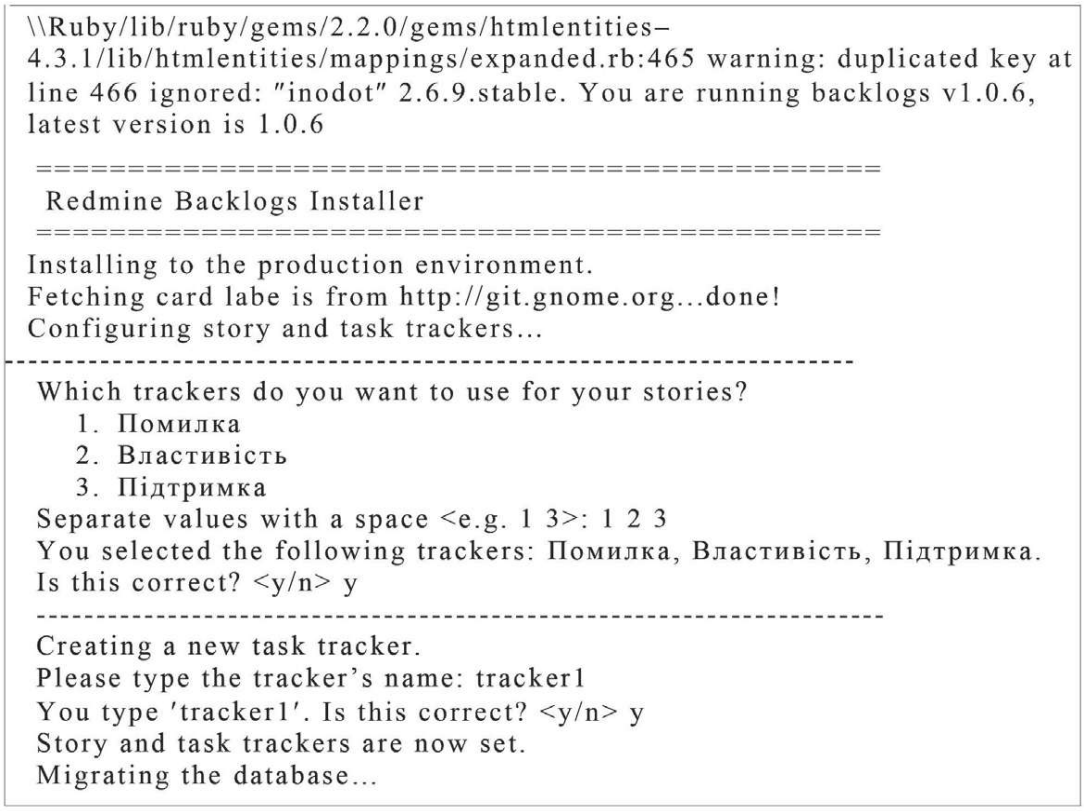


Рисунок 5.13 – Консольне вікно виконання команд встановлення плагіну Backlogs

Розгорнувши сервер Redmine як хмарний сервіс на незалежній обчислювальній платформі, можемо зробити деякі зауваження щодо особливостей його використання як хмарного Web-додатку:

1. Керування документами та файлами у сервері Redmine зводиться до операцій над ними, а саме: додавання, видалення та редагування їх. Проте правами доступу та маніпулюваннями над ними не можна отримати інформацію про маніпулювання з документами.
2. У сервері Redmine можна управляти лише правами доступу до завдань з розробки програмної системи тільки на рівні окремих інформаційних полів. Наприклад, зараз неможливо приховувати оцінку робочого часу над завданнями програмного проєкту від клієнтів. Проте є потенційна можливість зробити додаткові поля видимими лише користувачам з певними правами про файли.
3. У сервері Redmine немає такої можливості, як визначення коректної послідовності дій у workflow (що необхідно час від часу робити, змінюючи статус певних дій). Наприклад, зараз неможливо поставити вказівку про те, коли тестувальник закінчив виправляти помилку в програмному «таску». Отже, програмний менеджер має вибрати відповідального тестувальника та прикріпити до нього номер необ-

хідного «білда». Також неможливо приховати внутрішнє листування у межах програмного проєкту між учасниками групи розробників від замовника програмної системи.

4. У сервері Redmine у списку завдань неможливо задати загальну трудомісткість окремих задач програмного проекту.

5. Немає змоги задати користувачеві глобальну роль для всієї системи, наприклад «Керівник проєктного офісу», «Портфельний керівник офісу», які повинні мати доступ до усіх програмних проєктів, що зареєстровані у системі моніторингу управління проєктними розробками. Для цього потрібно виконати монотонну роботу, а саме додати відповідного користувача зі статусом «Керівник проєкту».

Можна виділити ще декілька інших проблем хмарного сервісу – сервера Redmine, зокрема незручне управління програмними завданнями із великою кількістю «тасків» (завдань), переміщення цих «тасків» між різними проєктами, пошуком в історії модифікацій завдань, фаховим налаштуванням типових оповіщень на e-mail тощо), проте ці проблеми не є критичними, їх легко виправити, і, ймовірно, такий функціонал з'явиться у сервері Redmine у найближчих релізах. А проблеми з профайлами та правами доступу і підтримкою ієрархічних зав'язків є наслідком недоопрацьованої архітектури сервера Redmine. Боротися з ними дуже складно, розглянувши необхідність у хмарних online-сервісах управління проєктною розробкою прикладних програмних систем, ми зупинились на розгортанні хмарного сервісу, а саме сервера Redmine, визначили особливості його розгортання для використання у спільній роботі над прикладними програмними проектами. Надали фахові рекомендації із встановлення цього сервера, як хмарного сервісу та його конфігурування для групової роботи, а також висвітлили певні зауваження щодо роботи із сервером Redmine під час адміністрування одного або декількох програмних проектів одночасно. Використання хмарних сервісів при розробці програмної системи забезпечило пришвидшення розробки та економію грошових коштів на придбанні ліцензійних програмних продуктів.

Склад команди розробників програмних систем в управлінні проєктами насамперед залежить від поставлених цілей та задач, які необхідно виконати. Відповідно до цього команду розробників програмного проєкту можна визначати відповідно до рівня організації та структури виконання поставлених завдань. Для кваліфікованого виконання поставлених завдань найперше необхідно виділити чотири

команди розробників програмних систем з використанням хмарної технології на незалежних обчислювальних платформах:

- команда інженерної підготовки процесу програмної розробки;
- команда з навчання зацікавлених осіб;
- незалежна команда тестувальників розробленої системи;
- незалежна команда із підтримки відповідної якості розробленої програмної системи.

На рівні управління розробкою програмної системи необхідно сформувати дев'ять команд, таких як:

- менеджери проєктної розробки;
- команда фахівців із керування конфігурацією розробленої системи;
- команда забезпечення якості програмної системи;
- команда програмістів;
- керівник із управління проєктом;
- проектувальники архітектури та структури програмної системи;
- команда із техніко-технічного забезпечення виробничого процесу;
- команда із внутрішнього та зовнішнього тестування програмної системи як частково, так і загалом;
- команда із розгортання та доменних аналітиків.

У сучасних парадигмах із розробки програмних систем з використанням хмарної технології на незалежних обчислювальних платформах широко використовуються методи глобальної розробки, які здебільшого віддалені. Тому у такому випадку необхідно зважати на специфіку формування таких команд із розроблення програмних систем та врахувати її. Кожна комунікація в середині команди з розроблення програмної системи має певну мету та ціль отримання отриманої інформації; висловлення альтернативних думок; перманентне навчання, адаптивна допомога або керівництво розробленою системою; підтвердження варіанту рішення поставленого завдання, підтримка у виконанні завдання, створення позитивного мікроклімату та соціального пакету; формалізація розпоряджень чи наказів, який відноситься до типів взаємодії у середині команди та визначається складом комунікантів команди.

Управління комунікаціями в середині команди з розробки програмної системи з використанням хмарної технології на незалежних обчислювальних платформах забезпечує підтримку системи обміну інформаційними потоками комунікативними засобами між членами

проекту, оперативну передачу управлінської та звітної інформації, яка спрямована на якісне забезпечення досягнення цілей програмного проекту. Кожен із членів команди розробників програмного проекту має бути фахово підготовлений до взаємодії між членами команди та підготовлений до виконання функціональних обов'язків.

Отже, технології або методи розподілу інформації між членами команд із розроблення програмних систем з використанням хмарної технології на незалежних обчислювальних платформах можуть мати значні відмінності залежно від якісних параметрів проєкту та програмних вимог систем автоматизованого контролю. Тому розглянемо основні компоненти створення програмних систем у вигляді UML-діаграм.

На нашу думку, хмарний сервіс, такий як сервер Redmine, є відмінним вибором сервісної програми для невеликих проєктів та команд, а також із навчальною метою. Для великих проєктів варто використовувати комерційні системи управління розробки програмних систем з використанням хмарної технології на незалежних обчислювальних платформах.

5.6. Декомпозиція програмних модулів системи на Sprint блоки

Для ефективного розроблення програмної системи з використанням хмарної технології на встановлений сервер Redmine як сервіс створюємо та записуємо спринти у яких визначаємо, які саме завдання має виконати той чи інший член команди розробників і за який час у залежності від його кваліфікації. Класично тривалість спринта визначається в один або два тижні у залежно від уподобань керівника проєкту. У межах того чи іншого спринта визначаємо задачі та під задачі із розроблення програмної системи, прикріпляємо виконавців, вказуємо вартість розроблення завдань, а також встановлюємо його пріоритет. Розкладаємо створені спринти залежно від технології процесу розроблення програмної системи з використанням хмарної технології на незалежних обчислювальних платформах на діаграмі Ґанта, врахувавши водночас тривалість виконання проєкту у загалом.

Для адміністрування розроблення програмної системи необхідно завантажити сервіс Redmine та створити в ньому спринти, відвідавши вкладку Backlogs, де ми додаємо Product Backlog спринти (рис. 5.14).

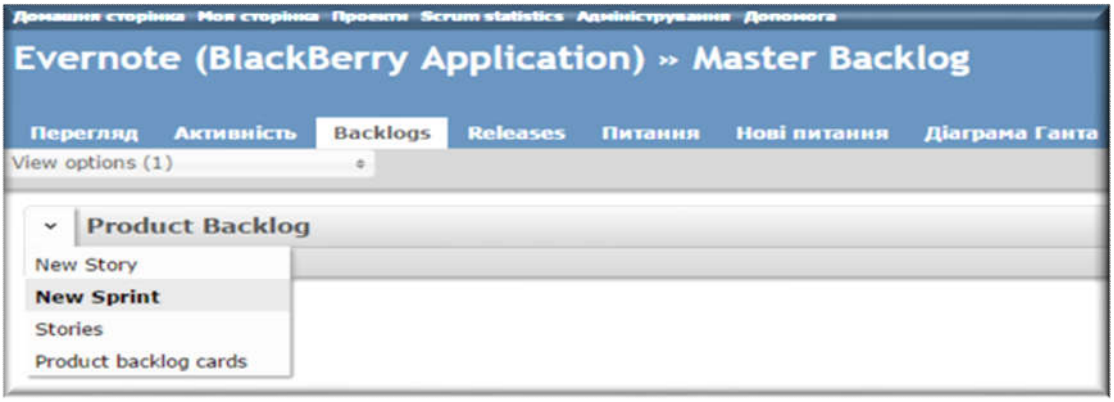


Рисунок 5.14 – Створення нових спринтів на хмарному сервісі Redmine

Для створення спринта з розроблення програмної системи необхідно вказати його назву та тривалість його виконання (рис. 5.15.).

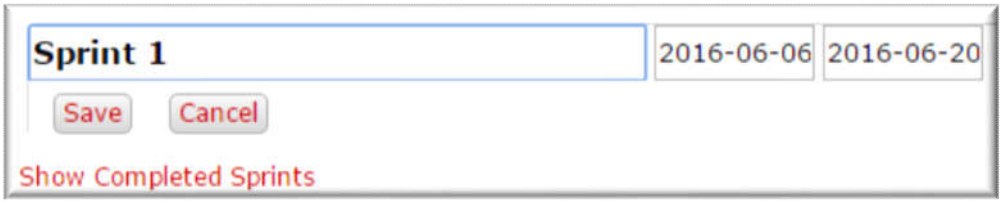


Рисунок 5.15 – Створення спринта із зазначенням його тривалості

Після того, як спринти було створено та занесено у хмарний сервіс Redmine для подальшого їхнього адміністрування додаємо всі User Story до спринтів (рис. 5.16).

Рисунок 5.16 – Історія документування виконання та модифікації спринтів

Розроблені та розподілені завдання у спринтах із визначенням пріоритетності виконання відображені у сервері Redmine. Кожний із них має визначений час у межах розробки програмної системи, а також декілька завдань, які члени команд із розроблення програмних

систем з використанням хмарної технології на незалежних обчислювальних платформах повинні виконати упродовж цього спринта (рис. 5.17).

Рисунок 5.17 – Виконання завдань із розроблення програмної системи згідно зі спринтами та пріоритетами

Створивши спринти та наповнивши їх завданнями для виконання членами команди з розроблення програмної системи і прикріпленими до них конкретного виконавця та час виконання завдань в межах вказаного спринта (рис. 5.18-5.21).

Sprint 1

Рисунок 5.18 – Деталізація завдань та виконавців за спринтом 1

Sprint 2

Рисунок 5.19 – Деталізація завдань та виконавців за спринтом 2

Sprint 3

Рисунок 5.20 – Деталізація завдань та виконавців за спринтом 3

Sprint 4

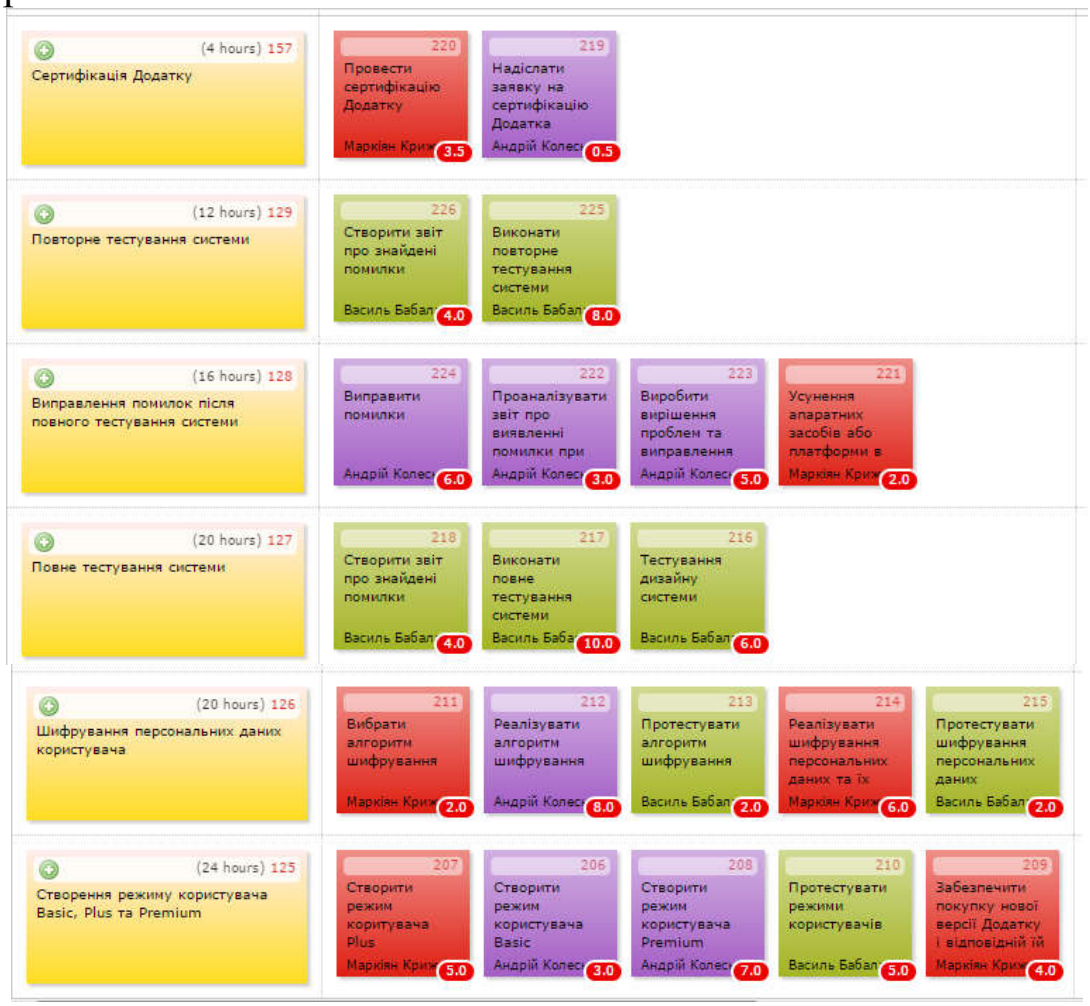


Рисунок 5.21 – Деталізація завдань та виконавців за спринтом 4

При розробленні спринтів ми розбиваємо розроблювальну програмну системи з використанням хмарної технології на логічні функціональні модулі із визначеним терміном виконання та прикріпленими членами команди з відповідною фаховою кваліфікацією. Що забезпечує якісне виконання поставлених завдань у казаний термін та визначеною послідовністю технологічних процесів у розробленні програмних систем засобами візуалізації, таких, як діаграми Ґанта. Дані діаграми у свою чергу забезпечують не тільки планові показники з розроблення програмних систем, але і забезпечують оперативний контроль виконання тих чи інших завдань спринтів.

У сервісі Redmine для всіх створених раніше завдань (Tasks), вказуємо дати фактичного їхнього початку та завершення, кількість витраченого на них часу та позначаємо, хто і коли їх виконує. Ще одним позитивним результатом такого відображення є визначення вартості завдання відповідно до термінів виконання та кваліфікації члена команди розробників. Це підтверджують наукові діаграми Ґанта (рис. 5.22-5.25).

Sprint 1

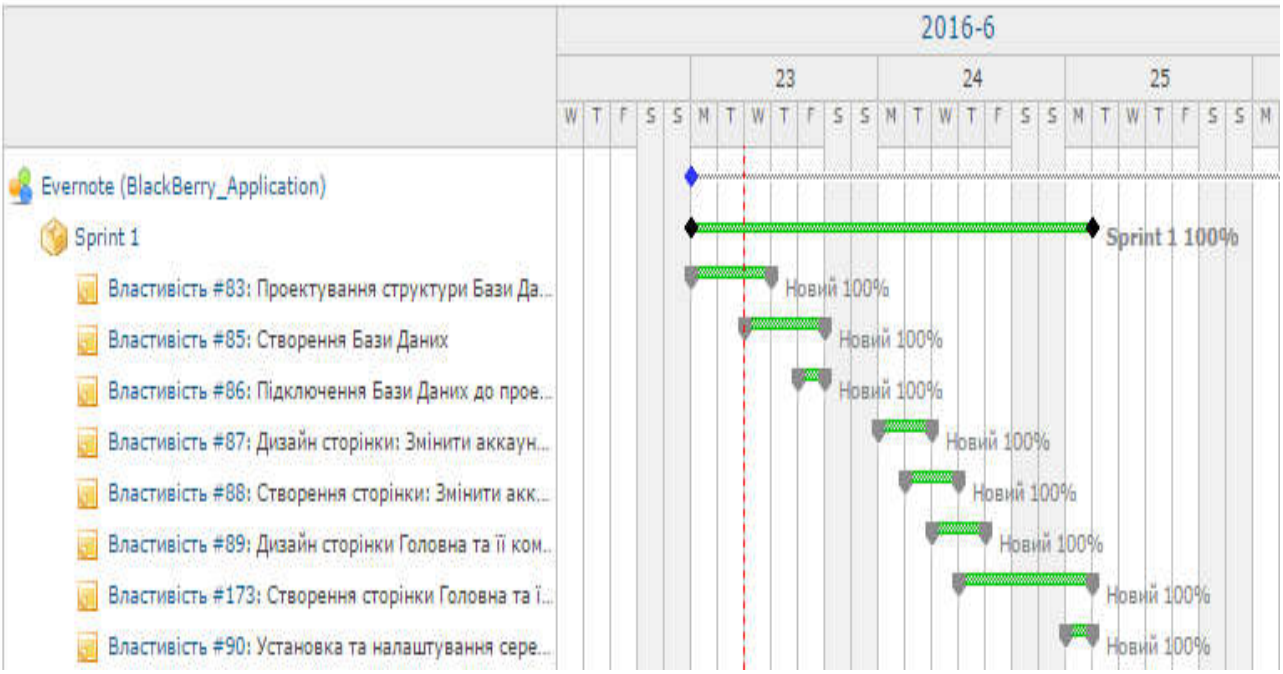


Рисунок 5.22 – Технологічний та часовий процес створення спринта 1 за діаграмою Ґанта

Sprint 2

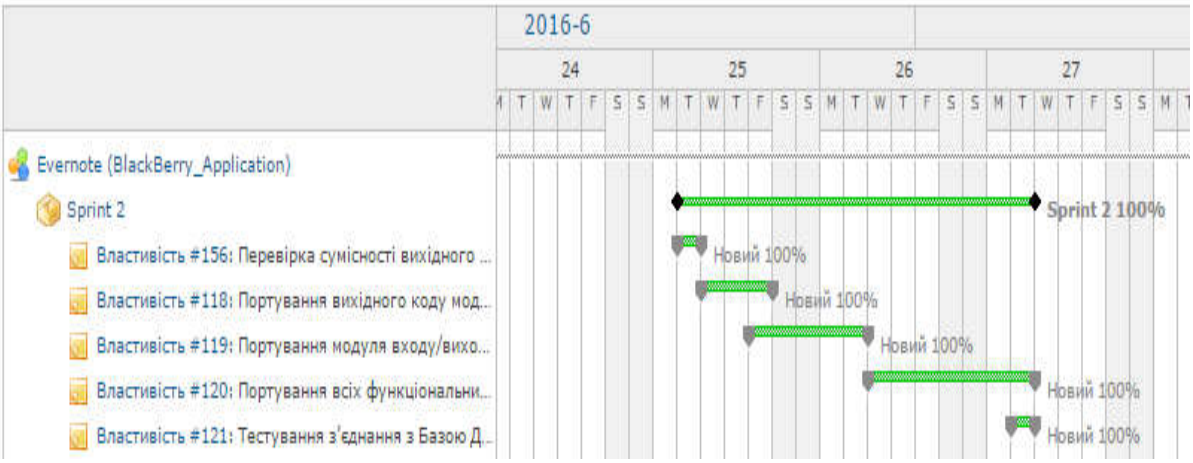


Рисунок 5.23 – Технологічний та часовий процес створення спринта 2 за діаграмою Ґанта

Sprint 3

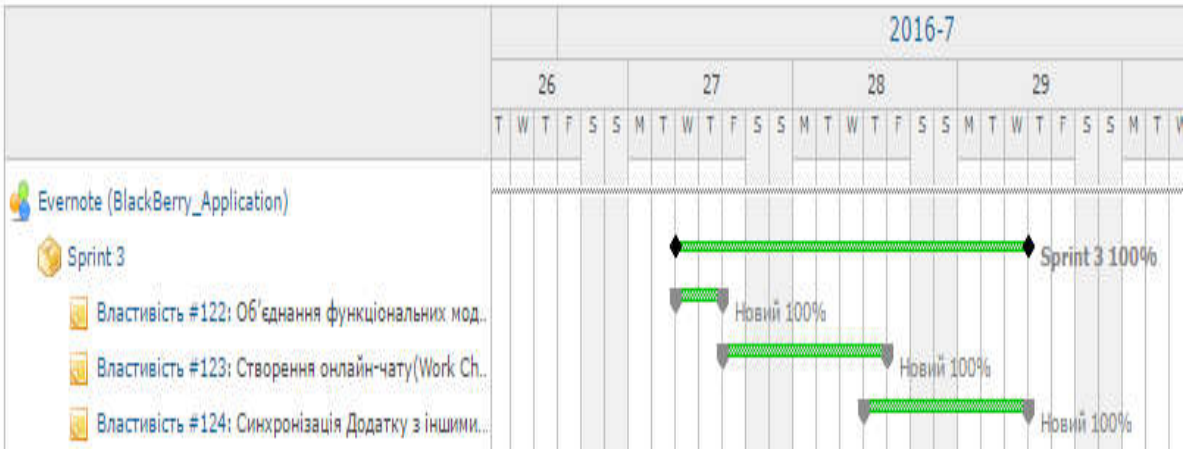


Рисунок 5.24 – Технологічний та часовий процес створення спринта 3 за діаграмою Ґанта

Sprint 4

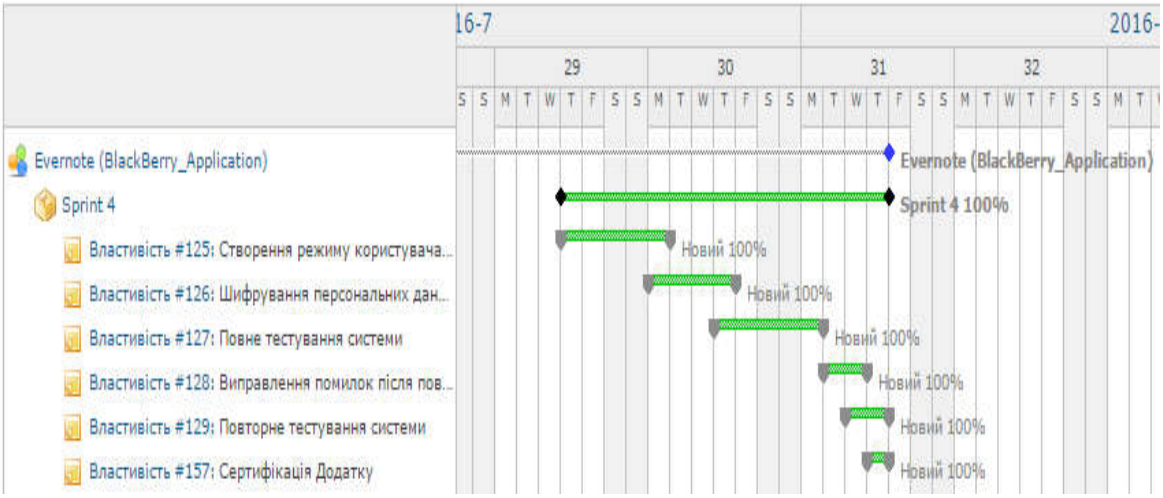


Рисунок 5.25 – Технологічний та часовий процес створення спринта 4 за діаграмою Ґанта

Проаналізувавши розроблення програмної системи з використанням хмарної технології на незалежних обчислювальних платформах засобами сервісу Redmine, ми відстежили перебіг та ітераційне виконання завдань проєкту. На основі представлених діаграм Ґанта можна зауважити, що всі спринти виконувалися достатньо чітко (без збурень у командах розробників), із мінімальними часовим відхиленнями від ідеальних планових значень, що у свою чергу дало незначне відхилення проєктного кошторису. Даний процес достатньо добре представлений на графіках Ґанта, де жоден спринт не був провальним, а якість результуючої програмної системи була на достатньо високому технічному рівні.

5.7. UML-візуалізація розробки окремих компонентів програмної системи з використанням хмарної технології

Візуалізація UML діаграм розробки окремих компонентів програмної системи відбувалася за допомогою хмарного сервісу draw.io, який розміщений на незалежній обчислювальній платформі та описує схеми взаємодії між модулями залежно від вимог проєкту та застосовуваних рекомендацій.

Створення UML-діаграми реєстрації користувача хмарним сервісом

Для початку проєктування програмної системи з використанням хмарної технології розробимо програмний модуль облікового запису користувача залежно від його категорії. Його створення облікового запису користувача відповідно до категорії повинне здійснюватися після встановлення програмного модуля на мобільний пристрій (рис. 5.26).

Створення UML-діаграми для входу користувача у систему

Користувач входить у розроблену програмну систему з використанням хмарної технології на незалежних обчислювальних платформах за допомогою раніше створеного облікового запису, виконуючи певну послідовність дій: користувач переходить у форму для авторизації; користувач вводить логін і пароль; система встановлює зв'язок із базою даних Evernote; система перевіряє коректність введених користувачем даних, порівнюючи їх з даними в Базі Даних Evernote; при коректному введенні логіна і пароля система здійснює перехід на робочу сторінку користувача або адміністратора відповідно до введе-

них даних; при неправильному введенні система виводить повідомлення з проханням перевірити правильність введення логіна та пароля і користувач вводить дані повторно (рис. 5.27).

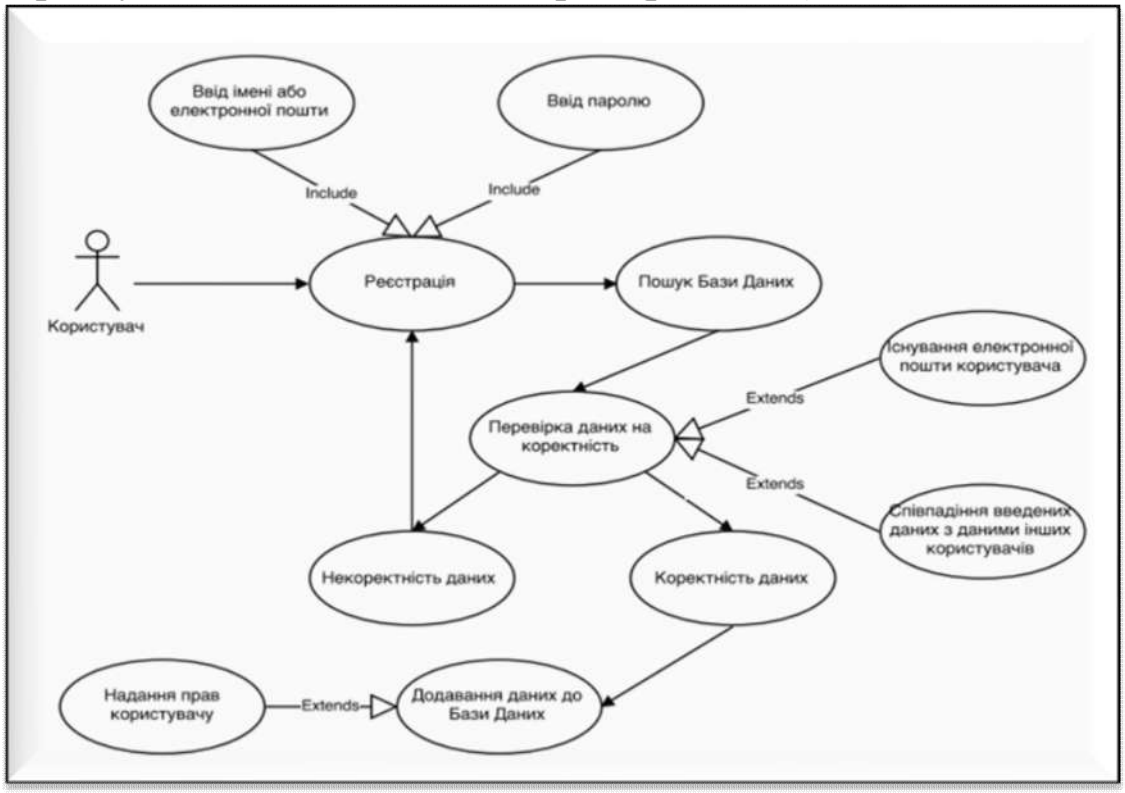


Рисунок 5.26 – Створення UML-діаграми реєстрації користувача

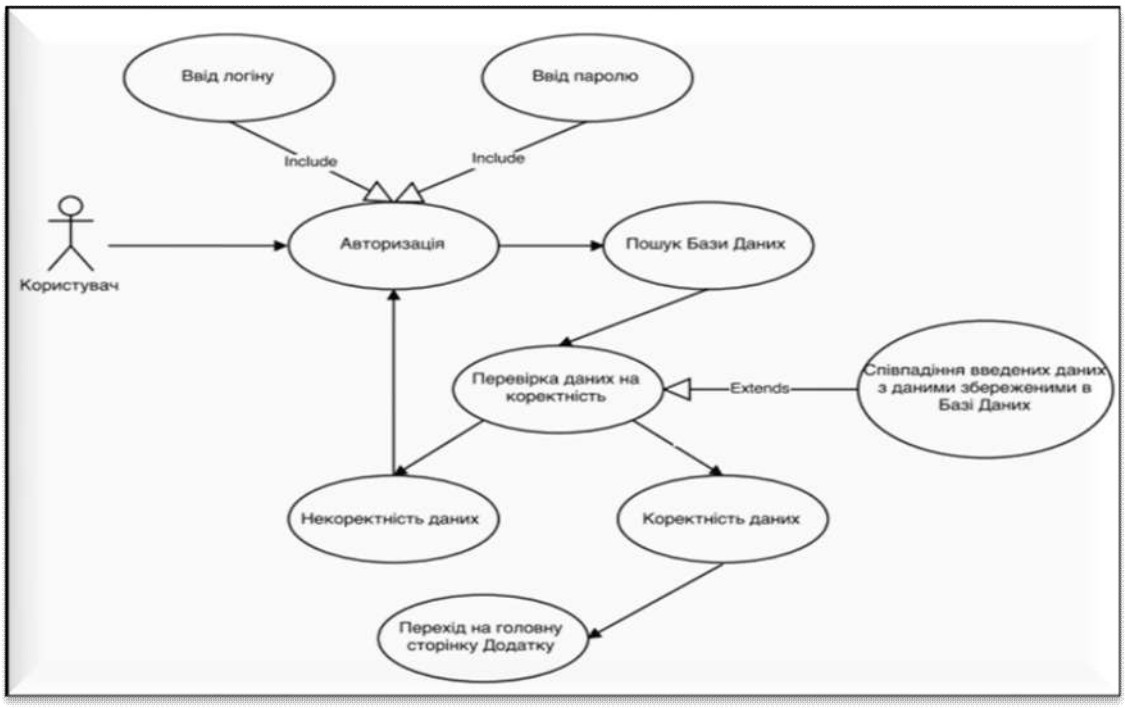


Рисунок 5.27 – UML-діаграма входу користувача у програмну систему

Створення UML діаграми основних компонентів програмної системи

Користувач створює всі доступні йому компоненти та описує послідовність дій: користувач авторизується у системі; за допомогою контекстного меню створює блокнот, вказуючи його назву; система здійснює перехід на сторінку з блокнотом та виводить його як елемент випадаючого списку з блокнота; користувач створює в даному блокноті нотатки; користувач двічі натискає лівою кнопкою мишки по нотатці, і система відкриває вікно з контекстним меню та полем для його редагування (запису текстової інформації, додавання зображень, аудіо, відео та інших файлів); користувач у даному вікні створює для поточного нотатку нагадування, вказуючи дату і час, коли треба повідомити користувача; користувач закриває вікно з нотатком, а система автоматично зберігає його та виводить у блокноті; користувач перетягує нотатки у вікно ярликів; користувач за допомогою контекстного меню додатку додає для нотатку мітку, вказуючи її назву; користувач синхронізує всі зміни з серверами Evernote, за наявності Інтернету; система автоматично синхронізує всі зміни з серверами Evernote, за наявності Інтернету (рис. 5.28).

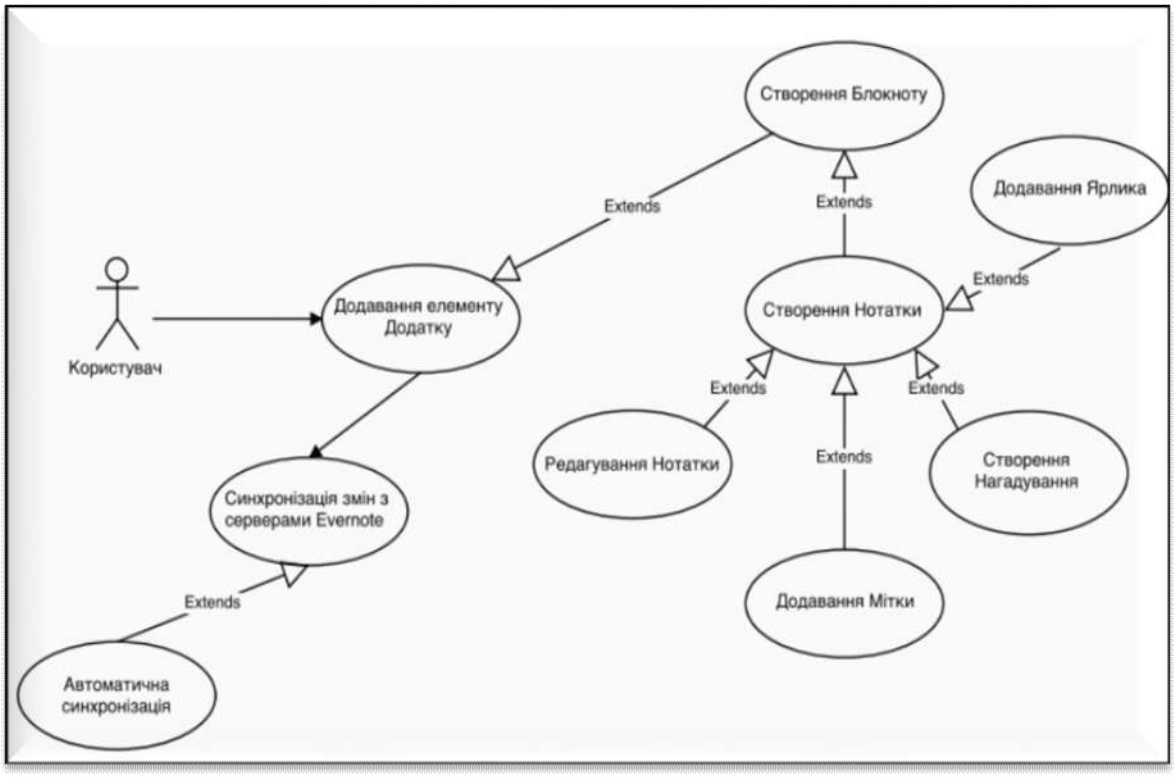


Рисунок 5.28 – UML-діаграма створення основних компонентів програмної системи

Створення UML-діаграми використання додаткового функціоналу

Користувач надає доступ до нотатки іншому члену команди з розробки програмних систем з використанням хмарної технології на незалежних обчислювальних платформах (рис. 5.29).

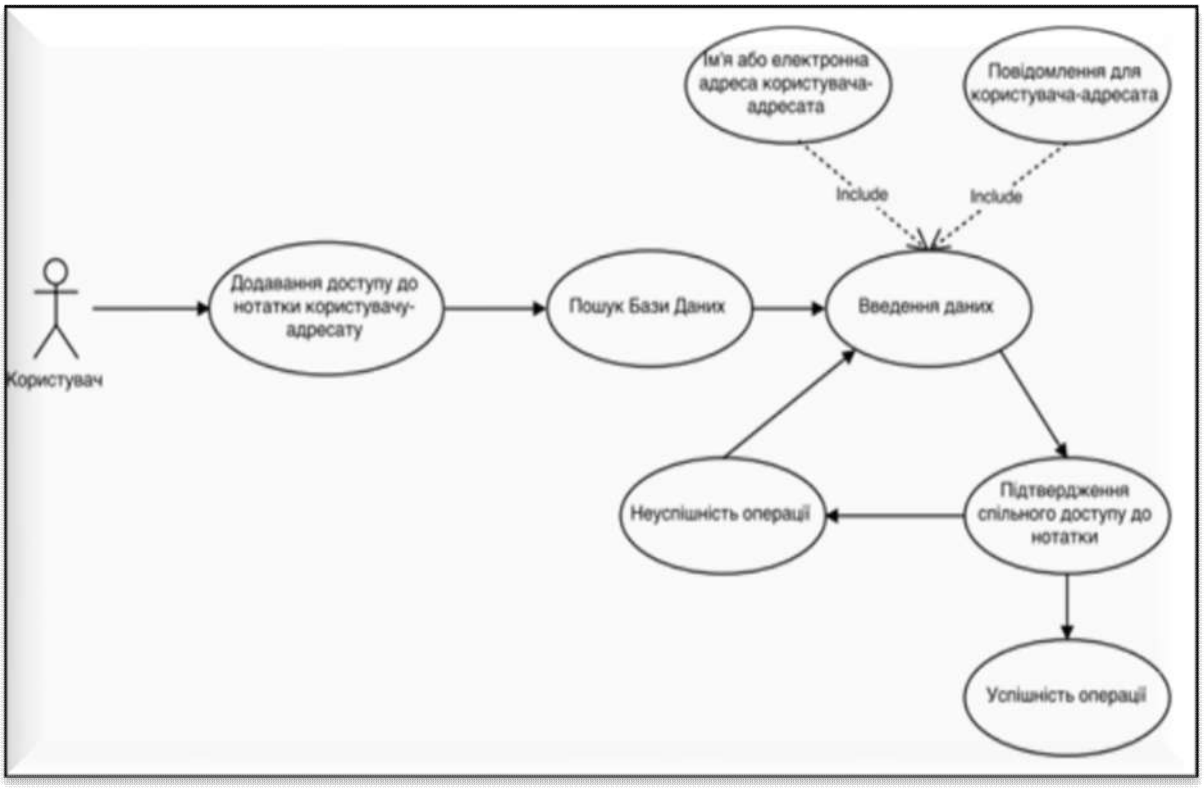


Рис. 5.29 – UML-діаграма створення додаткового функціонала програмної системи

Створення UML-діаграми нового інформаційного потоку

Для оперативного управління розробкою програмної системи з використанням хмарного сервісу, встановленого на незалежній обчислювальній платформі у вигляді сервера Redmine, членам команди необхідно постійно створювати нові інформаційні потоки обміну інформацією у вигляді нових розмов – Work Chat (рис. 5.30).

Створення UML-діаграми нового інформаційного потоку

Замовник хоче придбати оновлення програмного модуля до розширеної версії розгорнутої системи та отримати статус Plus-користувач або Premium-користувач (рис. 5.31).

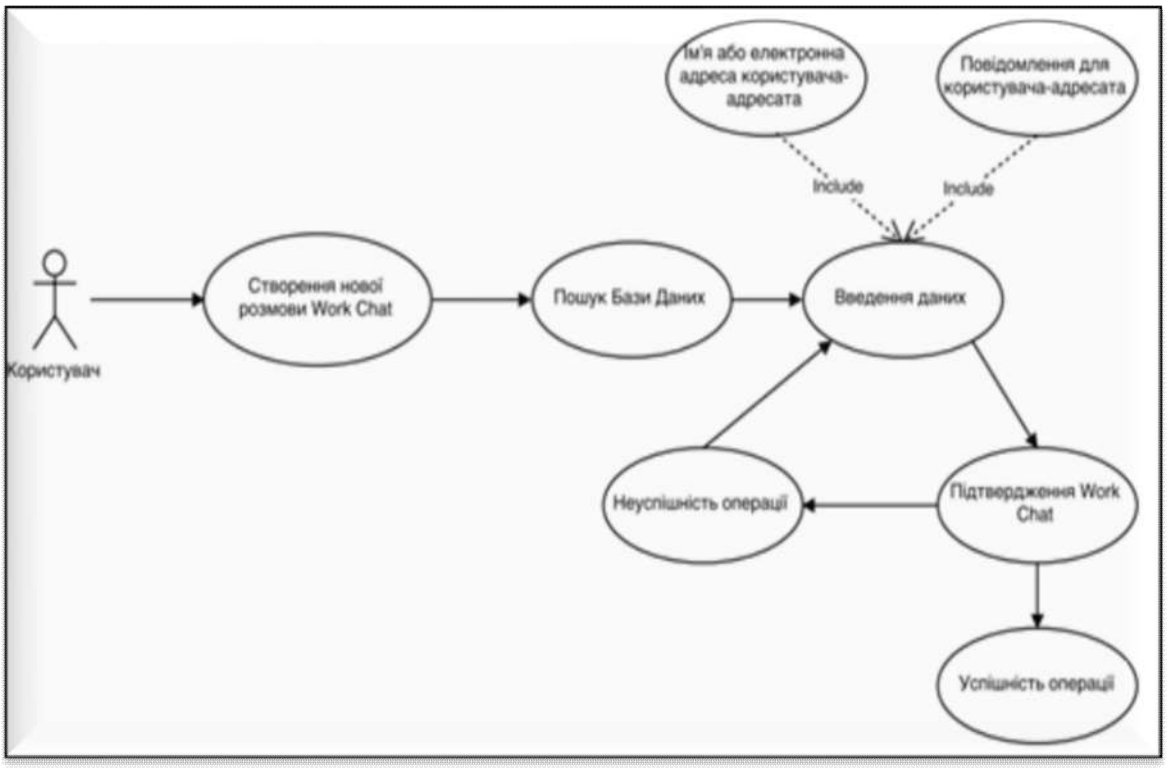


Рисунок 5.30 – UML-діаграма створення нового інформаційного потоку при оперативному управлінні розробкою програмної системи

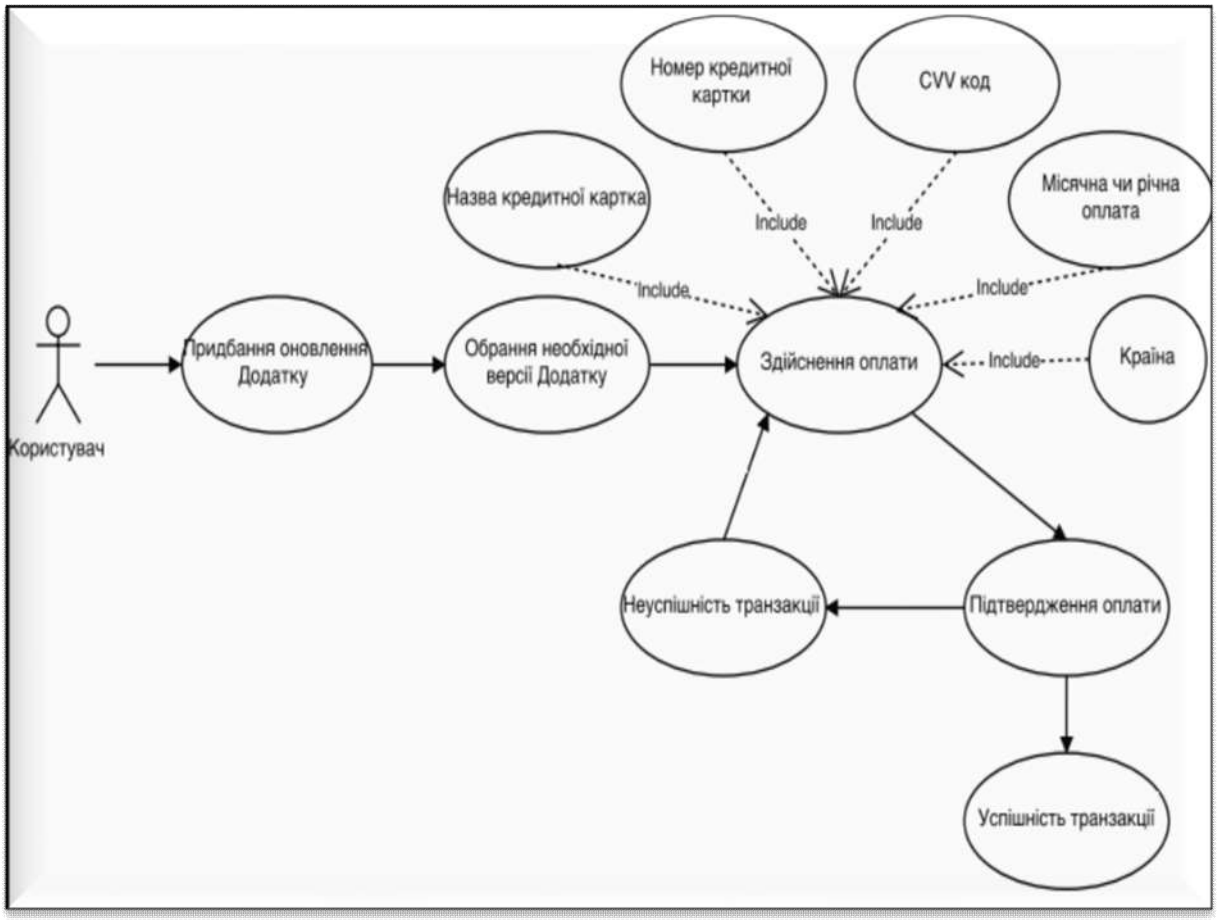


Рисунок 5.31 – UML-діаграма оновлення програмного модуля розробленої системи з використанням хмарної технології

Створення UML-діаграми Plus-користувача із наданням прав згідно з його категорією

У разі створення користувачів на стороні замовника з правами Plus та Premium вони набувають ряд додаткових можливостей, а саме Plus-користувач додає код блокування програмного модуля та підтверджує його повторним введенням, а також створює підказку, що виводитиметься у разі невірно введеного коду. При коректному введенні пароля система записує нові дані в базу даних Evernote, а при неправильному введенні система виводить повідомлення з проханням перевірити правильність пароля і користувач вводить дані повторно (рис. 5.32).

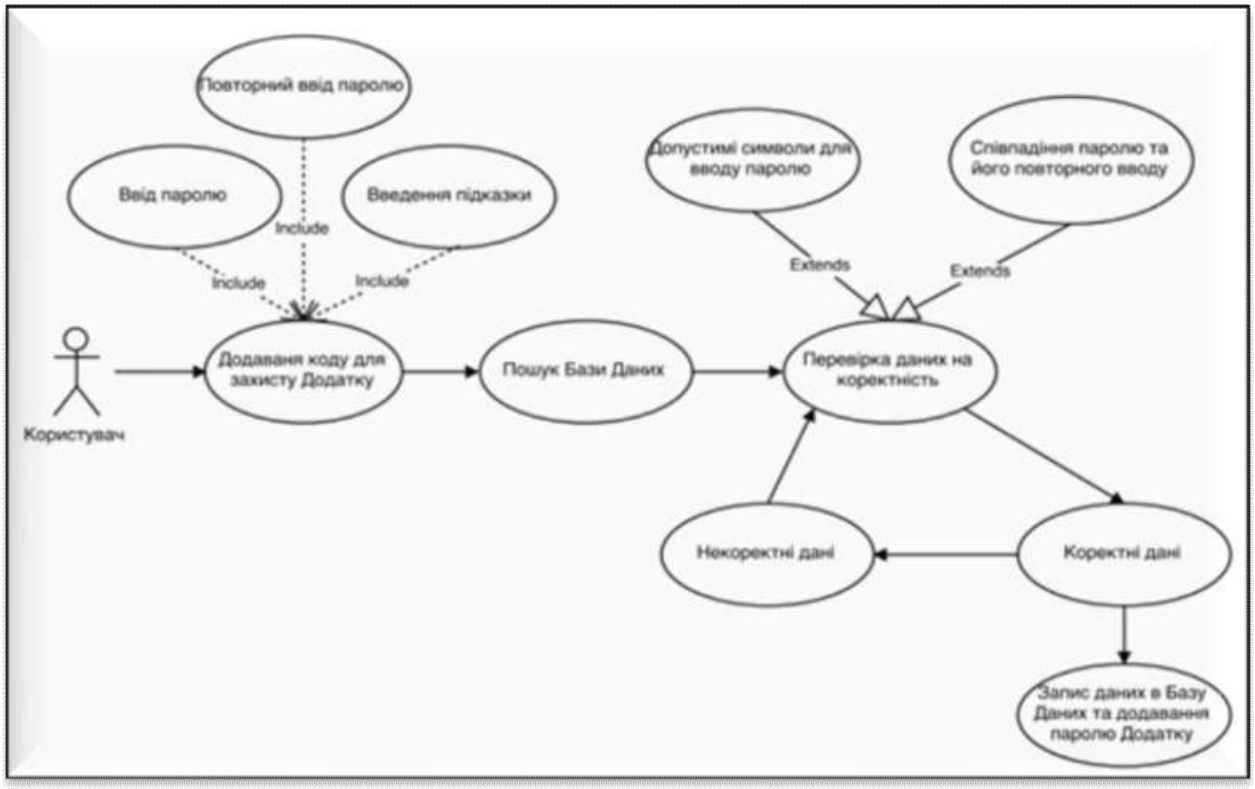


Рисунок 5.32 - UML-діаграма створення Plus-користувача із наданням прав згідно його категорії для захисту програмного модуля

Створення UML-діаграми Premium-користувача із наданням йому усіх прав згідно з його категорією.

При створенні Premium-користувача у програмному модулі розробленої системи з використанням хмарної технології на незалежних обчислювальних платформах йому стає доступний весь функціонал згідно з його категорії та прав (рис. 5.33).

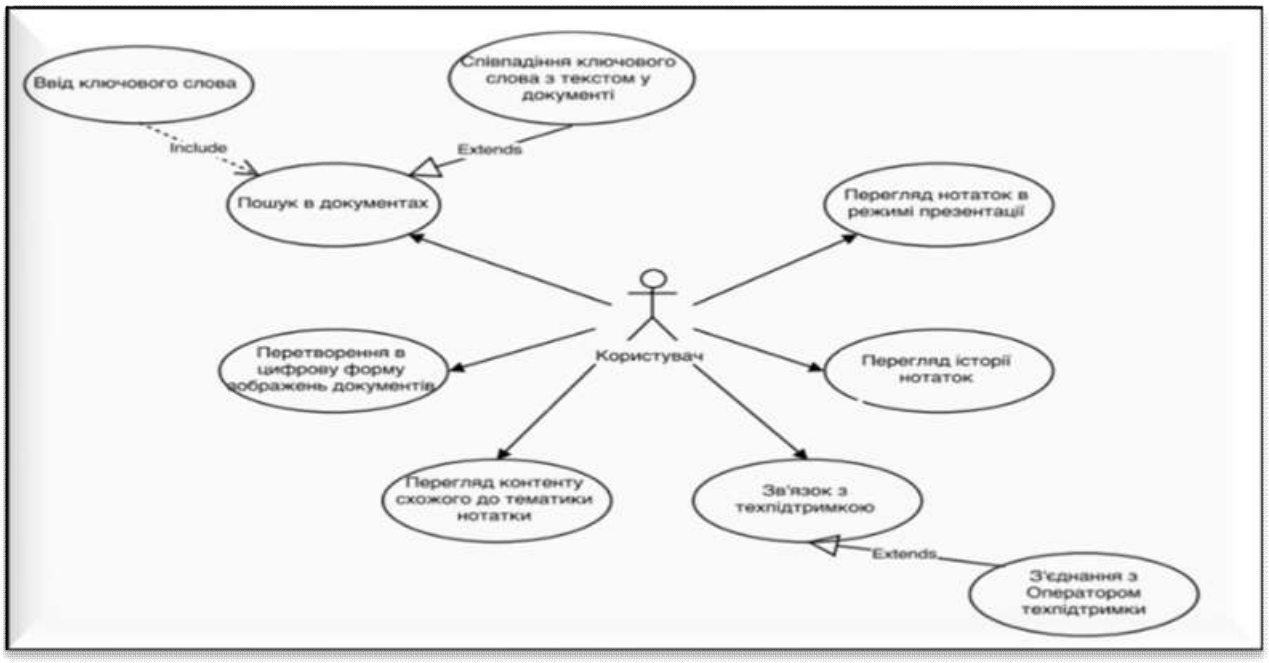


Рисунок 5.33 – UML-діаграма створення Premium-користувача із наданням йому усіх прав згідно з його категорією

Висновки до п’ятого розділу

Розглянуто та використано наукові підходи до хмарної технології під час розробки програмних систем, які ґрунтуються на емпіричних дослідженнях розподілення комп’ютерного навантаження на незалежних обчислювальних платформах через поділ (дроблення) сервісів на більшу кількість обчислювальних модулів. За такого адміністративного підходу вихід із ладу або значне завантаження одного із обчислювальних модулів на незалежній обчислювальній платформі є не настільки критичним, тоді коли його загальна функціональність мінімізована. Отже, запропонований науковий підхід значно збільшує сумарний обсяг вузлової та міжмодульної комунікації системи загалом. Тому його використання значно доцільніше для розподілених відмовостійких програмних систем, де оперативна втрата інформаційних даних просто неприпустима.

Запропоновані та вдосконалені архітектурні підходи використання хмарної технології на незалежних обчислювальних платформах забезпечують автоматизацію рівномірного розподілу обчислювального навантаження між вузлами, мінімізацію фінансових та часових втрат, оптимізацію надлишкових обчислювальних ресурсів, які можуть бути доступними у будь-який час і в будь-якому місці. Тому вдосконалений ефективний механізм оптимізації обчислювального навантаження у програмних модулях забезпечує мінімізацію втрат

неопрацьованих інформаційних потоків даних з хмарних ресурсів під час процедури їхнього перенесення та опрацювання. Отже, запропонований метод автоматичного відновлення працездатності програмної системи повинен визначати причину відмови того чи іншого модуля або вузла загалом.

Невпинне розширення використання хмарних технологій на теренах Всесвітньої мережі забезпечило створення різноманітних протоколів обміну інформаційними даними та сервісних інтерфейсів, що дало змогу розробляти програмні системи з використанням незалежних обчислювальних платформ програмно-апаратних засобів та сервісно-орієнтованої архітектури. Вдосконалено модель використання сервісно-орієнтованої архітектури для забезпечення зручного та якісного мережевого доступу до загального незалежного пулу конфігурованих обчислювальних сервісів.

Для забезпечення мінімізації фінансових та часових витрат при розробці програмних систем розгорнуто сервер Redmine як хмарний сервіс для менеджменту управління розробкою, що дало змогу отримати повноцінний доступ із будь-якого місця та у будь який час. Цей сервіс забезпечує методи розподілу інформації та прикріплення відповідних завдань до членів команд розробників програмних систем з використанням хмарної технології на незалежних обчислювальних платформах.

Для розробки основних компонентів програмних модулів у вигляді UML-діаграм використаємо незалежний хмарний сервіс draw.io, який розміщений на незалежній обчислювальній платформі, описує схеми взаємодії між модулями залежно від вимог проєкту та застосовуваних інформаційних технологій і рекомендацій.

РОЗДІЛ 6

ЗАСТОСУВАННЯ ТА ПРАКТИЧНА РЕАЛІЗАЦІЯ ГРУПОВОЇ ДИНАМІКИ ПРИ ФОРМУВАННІ ЕФЕКТИВНИХ КОМАНД РОЗРОБНИКІВ ПРОГРАМНИХ СИСТЕМ

6.1. Практична реалізація групової динаміки при формуванні команд

Розробка програмних систем з використанням хмарних технологій – це складний технологічний процес, який передбачає застосування різноманітної інформаційної технології, а також професійний добір висококваліфікованих членів у команди розробників.

Основну роль успішності програмного бізнес-проєкту з розробки програмних систем відіграє група професійних фахових працівників, яка сформована за методами групової динаміки, де враховуються такі параметри: розмір групи, розташування та розподіл повноважень, а також нормативні обмеження й вимоги управління. Сьогодні день не існує універсальної формули або технології для вирішення всіх потреб розробки програмних систем з використанням хмарної технології.

Динамічні групи розробників програмних систем працюють у різних умовах, і насправді кожна команда перебуває в унікальній ситуації. Суть проблеми є у тому, що розробники повинні адаптувати та застосувати всі свої когнітивні знання і підходи, щоб відобразити та подолати ризики, з якими може зіткнутися динамічна група при виконання поставлених завдань.

Результати аналізу дослідження цієї проблематики свідчать про те, що увага науковців акцентована на вивченні лише окремих аспектів професійного спілкування, здебільшого комунікативного. У сучасних дослідженнях мало приділяється уваги специфіці професійної діяльності інженерів із розроблення програмних систем в аспекті формування готовності до професійного спілкування.

Групова динаміка, ґрунтується на понятті групи розробників та комунікації всередині, визначає склад і взаємодію як членів груп між собою, так і між групами. До процесів групової динаміки належать: керівництво й лідерство, формування групової динаміки, згуртова-

ність групи, націленість групи на успіх, конфлікти і тощо. Через швидкі зміни в галузі інформаційних технологій та інших виробничих і не виробничих сферах обумовлюють потреби у розробці більш складних програмних систем і використанні для цього інноваційних методик. Як наслідок, ринок праці ставить перед майбутніми фахівцями якісно нові вимоги до професійної кваліфікації та особистісних характеристик.

З огляду на особливі аспекти розроблення програмних систем у діяльності спеціаліста з інформаційних технологій акцентується абстракція, моделювання, організація та подання інформації, управління змінами, а також реалізація проєкту і контроль якості, які в традиційному інженерному циклі зазвичай належать до фаз проєктування бізнес-процесу. Крім того, процес супроводу є критично важливим для життєвого циклу програмних систем.

Розробка програмних систем виконується для різних прикладних галузей, тому для фахівців з інформаційних технологій, які повинні володіти професійним досвідом та відповідною освітою у заданій предметній області, а також суміжних областями. Це дає їм змогу ефективно використовувати специфічні для предметної області методи, засоби і компоненти при розробці розподілених програмних систем з використанням хмарних технологій.

Професія розробника комп’ютерних програмних систем у сучасних реаліях є однією із популярних, але водночас на думку академіка А. П. Єршова, це одна з найважчих професій. Складність полягає у тому, що інженер-системотехнік або інженер-програміст повинен володіти здатністю першокласного математика до абстракції й логічного мислення у поєднанні з Едісонівським талантом споруджувати все, що завгодно, з нуля і одиниці. Він повинен поєднувати акуратність бухгалтера з провінційністю розвідника, фантазію автора детективних романів з тверезою практичністю економіста, а крім того, він повинен мати смак до колективної роботи, розуміти інтереси користувача і багато тощо.

Провівши системний аналіз та математичне моделювання при формуванні ефективного складу групи розробників програмних систем на методах групової динаміки з врахуванням психотипу кожного члена, ми визначили типові ситуації, з якими зараз стикаються команди.

Очікувано внаслідок аналізу з'ясувалося, що команди розробки програмних систем працюють у дуже широкому діапазоні ситуацій.

Залежно від бізнес-вимог розробники об'єднуються в невеликі команди – з десятьох членів або менше, середні команди розробників програмного забезпечення, які мають: 11-15 членів і дуже великі команди, складаються які мають понад 50 членів у команді.

Малі команди зазвичай перебувають в одному місці, проте використання сучасних комунікаційних технологій дає змогу створити команди, розподілених географічно тим або іншим способом. Деякі команди стикаються з відносно простими проблемами розвитку, але здебільшого командам доведеться долати значні технічні і технологічні труднощі в адаптації. Усі ці аспекти – склад команди, географічний розподіл, розподіл організаційних повноважень, технічної складності та нормативних проблем – мають великий вплив на те, як працюють команди і як вони організують себе, а в кінцевому етапі і на успішність виконання поставлених завдань.

З огляду на достатньо великий спектр проблем, які повинні вирішувати групи розробників програмних систем з використанням хмарної технології, розуміємо, що команди повинні бути гнучкими у своїх підходах. Команда з п'яти чоловік, які працюють над програмним проєктом будуть працювати по-іншому, ніж команда з 20 чоловік, які водночас працюють по-іншому, ніж команда з 200 чоловік. Команда, яка перебуває в одному офісі, буде працювати по-іншому, ніж команда, яка взаємодіє в різних офісах підходи якої відрізняються віт тих, які застосовують члени групи, що обирають віддалену форму роботи. Команди у різних ситуаціях потребують особливих організаційних структур, різних розподілів програмних модулів системи для забезпечення бізнес-процесу та відмінного забезпечення конфігураціями. Більшим групам потрібно більше професійного спілкування та значних організаційних витрат, отже, їхні члени ризикують більше ніж інші. Координація в межах невеликої Agile-команди організовується шляхом щоденних стендап-мітингів та детального планування розробки елементів комп'ютерної програмної системи, а також розподілу завдань між учасниками групи розробників. Для середніх команд може знадобитися два рівні координації розробників програмних систем – «stand-up meeting» або «scrum of scrum», а також мітинг детального планування з розподілом завдань всередині команди.

Типовий розподіл ролей у команді розробників програмних систем з використанням хмарної технології, що складаються з п'яти членів (мала типова група розробників), наведена на рис. 6.1, а на рис.

6.2 – типовий розподіл ролей у команді розробників програмних систем, що об’єднують від шести до десяти членів.

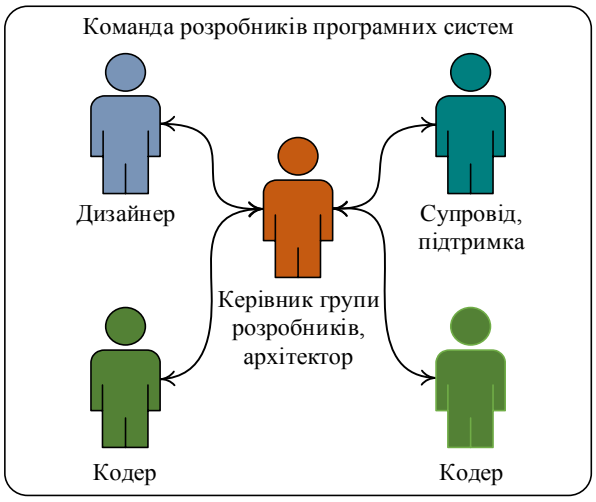


Рисунок 6.1 – Розподіл ролей у групах (до 5 членів)

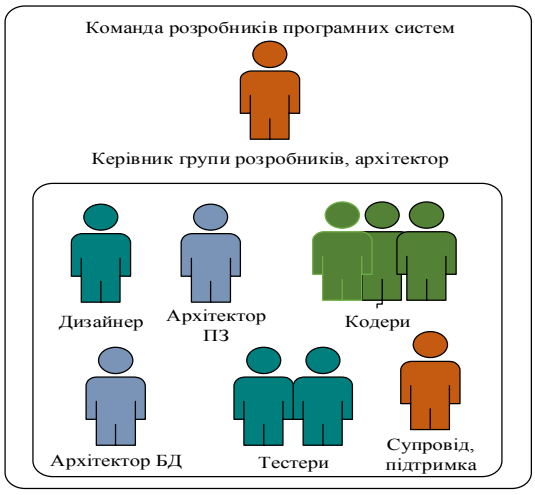


Рисунок 6.2 – Розподіл ролей у групі (від 6 до 10 членів)

Ефективність розробки програмної системи для бізнес-додатку залежно від кількісного складу групи наведено на рис. 6.3.

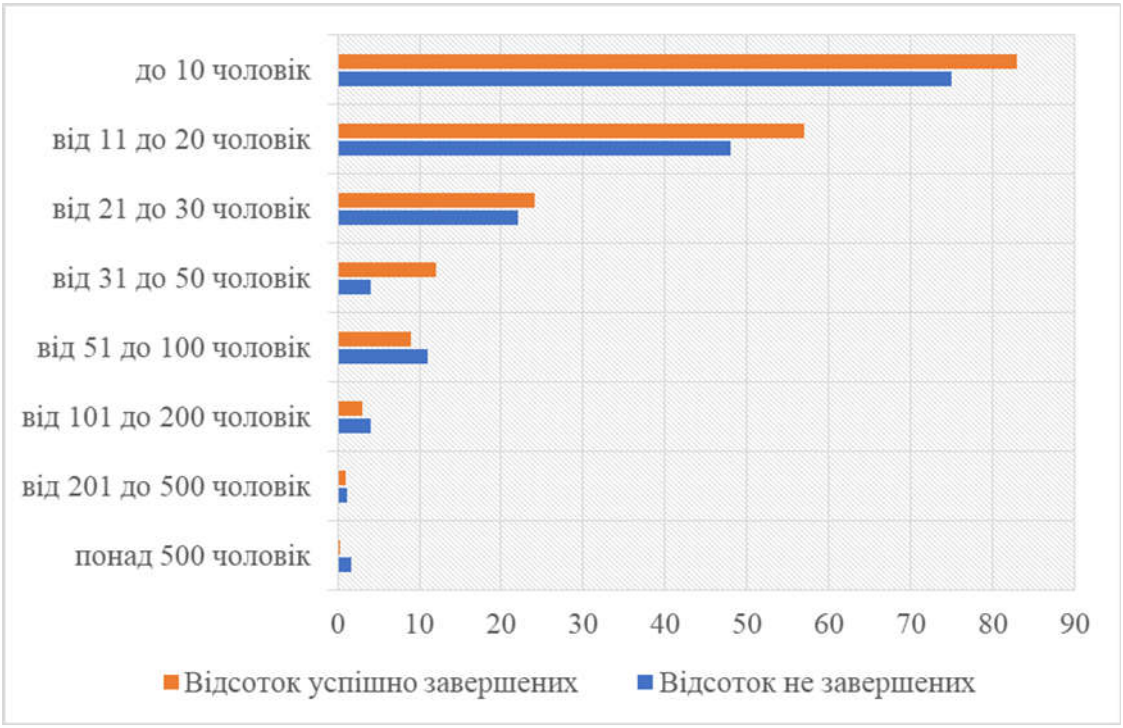


Рисунок. 6.3 – Динаміка успішних та не успішних розроблених програмних систем залежно від кількості членів команди

Отже, один шаблон не може бути універсальним, і не буде ніколи підходити. Упровадження більш гнучкої структури у команді розробників систем за допомогою архітектури підприємства й управління стратегіями в межах розвитку може ускладнити керування групою. Розроблення програмних Web-систем з використанням хмарної технології може бути непростою. Тому що, розроблення великої системи, організованої з п'яти підгруп або більше, вимагає набагато більш складних управлінських структур для вирішення архітектурної еволюції, управління вимогами та питаннями узгодження між групами. Складності координації стратегії збільшують ризики команди на фатальне завершення бізнес-проєкту, водночас ускладнюючи внутрішню організацію та взаєморозуміння у динамічній групі розробників програмних Web-систем.

Проведений системний аналіз у дослідженні показав, що використання технології Agile software development при розробленні програмних Web-систем є досить ефективною. Окрім того, використання методології групової динаміки разом із технологією Agile забезпечує вищий моральний дух команди, що загалом покращує продуктивність праці та внутрішньої дисципліни персоналу в команді розробників програмних
Web-систем.

6.2. Вплив групових ефектів на формування успішних команд розробників програмних Web-систем

У результату наукового дослідження питань з розроблення програмних Web-систем з використанням хмарної технології опрацювання інформаційних потоків даних на незалежних обчислювальних платформах досліджено, що діяльність членів команди в області програмної інженерії має свою особливість. Зазвичай усі члени команди розробників програмних Web-систем, які працюють у сфері інформаційних технологій – «інтроверти». Тобто це спеціалісти, які зосереджені на своєму внутрішньому світі, черпають свою енергію і витрачають її на події усередині себе. Їм переважно потрібна самота для отримання та опрацювання інформаційних потоків даних. Вони насамперед прагнуть створити враження про об'єкт опрацювання і тільки потім взаємодіяти з ним. Водночас не потребують широких про-

фесійних контактів, хоча зовнішньо можуть і не мати проблем у спілкуванні.

Відбір та формування команди фахівців відбувається у продовж цього розроблення програмних Web-систем і суттєво впливають на нього, тому у команді цінують як спеціальні інженерні навики, пов'язані з розробкою програмних Web-систем, так і загальнолюдські якості, які під впливом специфіки професійної діяльності починають виступати на перший план як професійні. Через варіативні грані свого психотипу члени команд розробників програмних Web-систем реагують по-різному на своє професійне оточення та мікроклімат у ньому. Тому найбільш важливим на початковому етапі створення команд розробників програмних Web-систем з використанням хмарної технології є визначення психотипів кожного із членів з подальшим узгодженням (врахуванням) цих психотипів при формуванні команди з метою успішного виконання поставлених завдань та мінімізацією внутрішньо командних збурень. Досліджуючи психотипи членів команд розробників програмних Webсистем та їхній вплив на успішне завершення поставлених завдань, ми намагаємося зрозуміти і пояснити, який вплив здійснюється на думки, почуття і поведінку членів команд: дійсна, уявна або передбачувана присутність інших.

Реакція людини на присутність інших проявляється у вигляді достатньо широкого різноманіття групових ефектів. Використання групових ефектів як механізму функціонування групи розробників програмних Web-систем, здійснюється динаміка групових процесів всередині команди та досягається внутрішньогрупові позитивні стани. Вони є консолідованими засобами, що забезпечують когнітивну інтеграцію індивідуальних дій у спільній груповій діяльності та спілкуванні як всередині групи розробників, так і за її межами. До переліку групових ефектів при формуванні груп розробників програмних систем з використанням хмарної технології залучають такі ефекти: соціальної фасилітації; належності до групи; Рінгельмана; «синергії»; групового мислення; конформізму; моди (наслідування); «ореолу»; групового фаворитизму; групового егоїзму; «маятника»; «хвилі»; «пульсара»; «бумеранга»; «ми і вони».

Члени команди розробників програмних Web-систем з використанням хмарної технології для опрацювання інформаційних потоків даних на незалежних обчислювальних платформах (вузлах), які об’єднуються у зацікавлені групи, неминуче зіштовхуються з груповими ефектами. Розглянувши детальніше формування групи розроб-

ників програмних Web–систем, з якими ефектами вони можуть зіштовхнутися під час свого професійного існування наведено на (рис. 6.7).

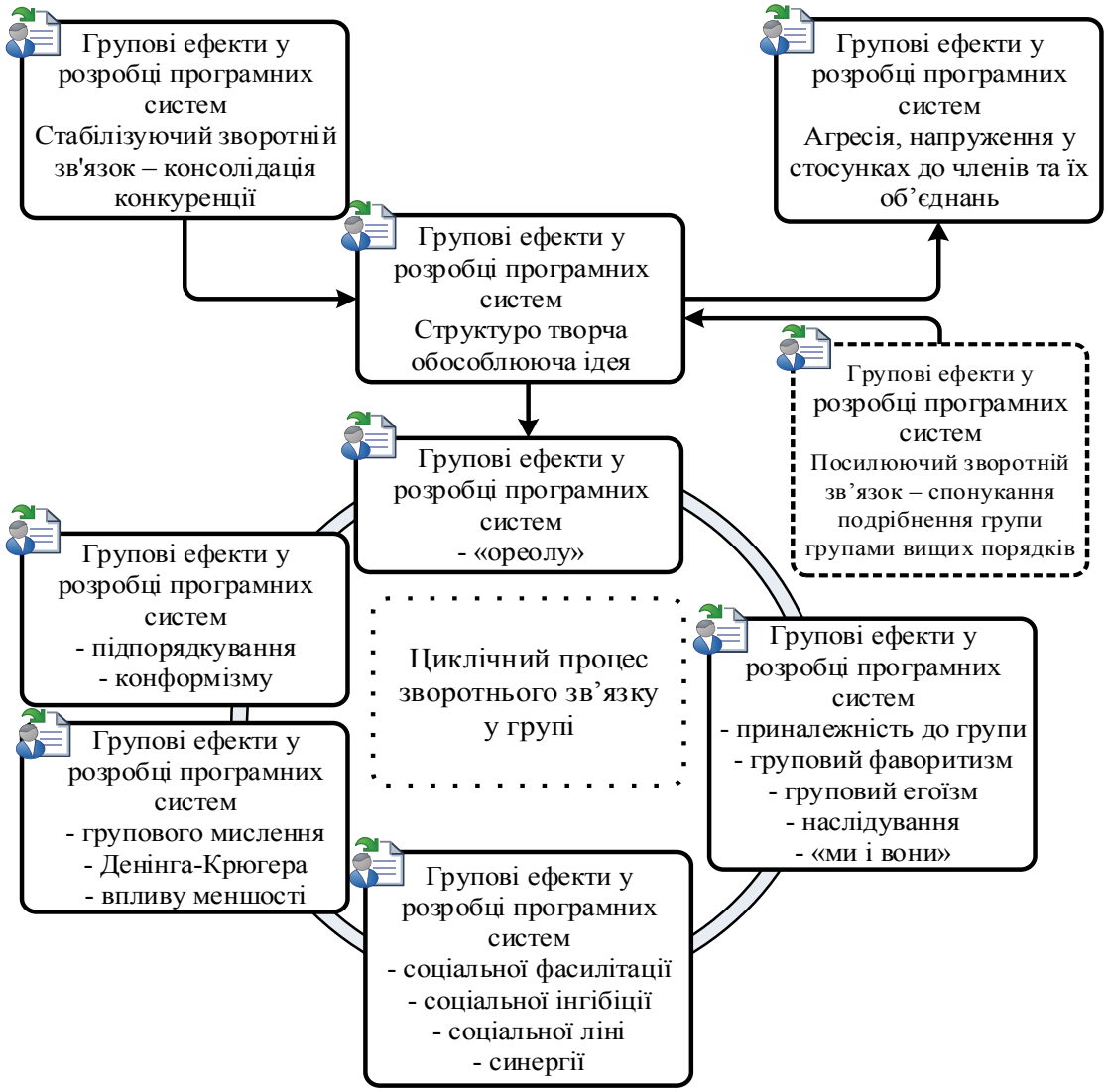


Рисунок 6.7 – Модель структури групових ефектів та взаємодія між ними при розробленні програмних систем

Для того, щоб сформувати ефективну групу розробників програмних Web-систем з використанням хмарної технології, необхідне завдання та керівник, так званий менеджер проєкту, який зможе збалансувати групу відповідно до фахових компетенцій кожного із членів з врахуванням їх психотипів та скерувати її у правильному робочому руслі для виконання поставлених завдань. Менеджер повинен правильно зрозуміти, що очікує замовник і правильно розподілити надані ресурси (команда розробників програмної системи, матеріальний та часовий ресурс).

Звернемо увагу на ефект Рінгельмана, який тлумачиться наступним чином: чим більше членів команди розробників програмних Web-систем з використанням хмарної технології бере участь у вирішенні поставлених обсягів завдань, тим буде менший їхній сумарний (когнітивний) внесок, ніж кожного окремого члена команди. Також відзначимо, що розроблення програмної Web-системи – це достатньо складне та тривале завдання, і його варто розділити на менші, але результативні завдання. З огляду розуміємо, що на виконання таких подрібнених завдань варто залучати не більше двох-трьох членів команди розробників програмних Web-систем з використанням хмарної технології.

Також одним із важливих ефектів є ефект синергії. Він чітко проявляється, коли двоє або декілька членів команд розробників виконують одне просте завдання, яке потребує знаходження різних способів без критичного аналізу і логічного усвідомлення їхніх наслідків. Синергія у груповому мисленні проявляється і під час вирішення усією командою розробників програмних Web-систем або більшою її частиною, досягаючи при цьому певного способу вирішення поставлених завдань або подолання перешкод, які виникли у процесі вирішення завдань на шляху до успішного їх виконання. Отже, при виникненні певних виробничих або соціальних проблем у команді розробників програмних Web-систем варто обговорювати методи їхнього подолання у колективі.

Залучаючи недосвідчених розробників до розроблення програмних Web-систем з використанням хмарної технології, менеджер повинен стежити за проявом ефекту конформізму. Новий член команди, якого долучений до виконання завдань проєкту, може подавати досить цікаві, інноваційні, нетривіальні ідеї, але груповий тиск, а також впертість та наполегливість у поведінці інших змушують новачка відмовитись від своїх думок і вподобань, а також фахових компетенцій та переорієнтовуватися на думки інших.

До цієї групи ефектів можемо зарахувати ще один схожий ефект групового егоїзму, який частково перетинається з ефектом конформізму. Обираючи часові межі для виконання поставлених завдань із розробки програмної Web-системи, варто зважати на ефект маятника, на який також мають звертати увагу не тільки керівники проектів, а й усі інші учасники команди. Даний ефект ґрунтується на вмінні професійно організовувати як робочий час, так і відпочинок (зокрема і в ви-

гляді соціального пакету), з огляду на емоційний стан та дисципліну в команді.

При поширенні нових інформаційних потоків даних та нової цікавої інформації для команди відбувається ефект хвилі. Коли один із членів команди розробників програмних Web-систем знаходить новий цікавий метод вирішення тієї чи іншої виробничої ситуації, пропонує його команді, то цей потік корисної інформації поширюється у всій команді, наче хвиля. Отже, такі інформаційні повідомлення можна розцінювати як позитивний ефект у команді розробників програмних Web-систем з використанням хмарної технології з опрацювання інформаційних потоків даних на незалежних обчислювальних платформах .

З появою актуальних програмних завдань при роботі над проєктом проявляється ефект пульсара як готовність команди до групової взаємодії та активності. Він полягає у стрімкому підвищенні активності кожного члена команди розробників на початку виконання поставлених завдань, а коли проблемне завдання вирішене, то настає суттєве зниження активності. Потім групова динаміка активності членів повертається на оптимальний рівень, який необхідний для ефективної, злагодженої та безупинної роботи команди з розроблення програмних систем.

Отже, члени команди розробників програмних систем з використанням хмарної технології на початковому етапі повинні подолати внутрішні суперечності, пройти через конфлікти перш, ніж сформується дійсно професійний колектив, на позитивне виконання поставлених програмних завдань зі створеними загальнокомандними цілями. Можлива внутрішня професійна конкуренція, суперництво, суперечки щодо своїх уподобань, а також оборонна позиція. Неодмінні складнощі або невдачі у процесі створення програмної Web-системи можуть породжувати конфлікти, «пошук винних».

Лідеру у команді на цьому етапі вкрай важливо забезпечити відкриту продуктивну комунікацію у команді – конфлікти не потрібно приховувати або вирішувати різко. До внутрішніх суперечок, які відбуваються у середині команди розробників програмних систем, необхідно ставитися толерантно, виважено, спокійно, терпляче та ретельно продумувати вихід із ситуації, яка зараз. Щоб змінити самоствердження та агресивні прояви амбіцій, обов’язково прийде продуктивна фахова співпраця членів команд розробників. Чіткий розподіл завдань у межах команди розробників забезпечить уникнення дублю-

вання функціональних обов’язків. Лідер перестає перебувати у стані постійного стресу при виконанні членами команд поставлених програмних завдань з розроблення Web-систем. Отже, робота по формуванні команди розробників програмних систем з використанням хмарної технології на цьому етапі – це вже не екстремальне вирішення конфліктних ситуацій, а скрупульозна і наполеглива праця по відпрацюванню загальних норм та правил співіснування членів команди.

Обов'язково у команді мають бути чітко сформульовані та впроваджені певні механізми ефективної оцінки та перегляду внутрішніх правил, які на даний час уже втратили свою актуальність або ефективність. Настає той професійно виважений етап, коли команда розробників починає працювати значно ефективніше та відчувати високий командний дух (професійну згуртованість), коли члени команди розробників програмних Web-систем добре знають та розуміють один одного й уміють використати сильні професійні сторони колег. На цьому етапі забезпечується високий командний рівень довіри при функціонуванні цієї команди. Це один із найкращих періодів для розкриття індивідуальних фахових компетенцій членів команд розробників програмних систем з використанням хмарної технології. Члени команди постійно прагнуть різних можливостей для удосконалення своїх фахових компетенцій і знаходять, проте вони всього зацікавлені у кар’єрно-професійному зростанні.

Якщо ж у процесі розробки програмної Web-системи внутрішній мікроклімат команди так і не склався, то неминуче розформовування команди. Зазвичай це трапляється через допрацювання керівника проєкту. Одна із головних задач у такому випадку перевірити та передати замовнику результати розробки цього проєкту, а також максимально акумулювати отриманий як позитивний, так і негативний досвід розроблення програмних Web-систем з використанням хмарної технології.

6.3. Визначення та врахування психотипів кандидатів у команди розробників програмних систем

Для визначення психотипів претендентів у команди розробників програмних систем з використанням хмарної технології було використано автоматизовану комп’ютерну програму. Вона складалася з 81 питання, на які претенденти у члени команд розробників програмних

Web-систем мали дати однозначну відповідь – «Так» або «Ні». Такий перелік подано у Додатку А.

Після проходження цього тесту ми визначали психотип кожного із претендентів у команду розробників програмних систем з використанням хмарної технології. На основі цих результатів у нас з'явилася можливість якісно підібрати членів команд розробників у такий спосіб, щоб у кожну команду потрапляв тільки один член із лідерським психотипом. Такий підхід забезпечив нам максимальну керованість групою розробників програмних Web-систем. Наступним кроком у проведенні нашого експерименту була побудова діаграми фахового рівня знань із представленням психотипів кожного із члені в розробників програмних систем з використанням хмарної технології за допомогою програмного модуля 6.1.

Програмний модуль 6.1.

```
#!/usr/bin/env python
import math
class Ellipse:
 def __init__(self, x,y, rx,ry):
  self.x  = x
  self.y  = y
  self.rx = rx
  self.ry = ry
class Matrix:
 def __init__(self, ellipse):
  self.half_width  = max(int(math.ceil(ellipse.rx + el-
lipse.x) + 1),
                         int(math.ceil(ellipse.ry + el-
lipse.y) + 1))
  self.half_height = self.half_width
  self.width       = self.half_width * 2
  self.height      = self.half_height * 2
  self.cells       = [[0 for x in range(self.width)] for y in
range(self.height)]
def display(self):
  codes = '0123456789abcdefghijklmnopqrstuvwxyz'
```

```
  print('\n'.join([''.join([codes[self.cells[y][x]] for x in
range(self.width)])
                  for y in range(self.height)]))
 def draw_ellipse(self, i_bit, ellipse, angle):
  mask_bit  = (1 << i_bit)
  sin_angle = math.sin(math.radians(angle))
  cos_angle = math.cos(math.radians(angle))
  for matrix_y in range(self.height):
   for matrix_x in range(self.width):
    cell_x_unrotated = matrix_x - self.half_width
    cell_y_unrotated = matrix_y - self.half_height
    cell_x = sin_angle * cell_y_unrotated + cos_angle *
cell_x_unrotated
    cell_y = cos_angle * cell_y_unrotated - sin_angle *
cell_x_unrotated
    cell_dx = float(cell_x - ellipse.x) / ellipse.rx
    cell_dy = float(cell_y - ellipse.y) / ellipse.ry
    # print('%3d%3d | %4d%4d | %6.2f%6.2f | %6.2f %d' %
    #       (matrix_x,matrix_y, cell_x,cell_y,
cell_dx,cell_dy,
    #        cell_dx * cell_dx + cell_dy * cell_dy,cell_dx *
cell_dx + cell_dy * cell_dy > 1))
    if (cell_dx * cell_dx + cell_dy * cell_dy <= 1):
self.cells[matrix_y][matrix_x] |= mask_bit
def draw_ellipses(self, ellipse):
  for i_ellipse in range(5): self.draw_ellipse(i_ellipse, el-
lipse, i_ellipse * 72)
def count_regions(self):
  counts = [0 for code in range(1 << 5)]
  for matrix_y in range(self.height):
   for matrix_x in range(self.width):
    counts[self.cells[matrix_y][matrix_x]] += 1
  area_all  = self.width * self.height
  area_min  = area_all
  area_non0 = area_all - counts[0]
  for i_count in range(len(counts)):
   if (area_min > counts[i_count]): area_min =
counts[i_count]
```

```
  if (True):
   # if (area_min > 0):
    bar = ''.join(['%-5d+----' % i for i in
range(99)]).replace(' ','-')[1:]
    ppm = 1e6 * area_min / area_non0
    print('x:%2d y:%2d rx:%2d ry:%2d ppm:%4.0f %s' %
      (ellipse.x,ellipse.y, ellipse.rx,ellipse.ry, ppm,
bar[:int(ppm / 100 + 0.5)]))
ellipse = Ellipse(9,14, 40,80)
ellipse = Ellipse(7,16, 42,78)
ellipse = Ellipse(5,12, 46,70)
ellipse = Ellipse(5,14, 37,64)
ellipse = Ellipse(5,12, 40,65)
ellipse = Ellipse(5,12, 41,66)
ellipse = Ellipse(10,24, 82,132)
matrix  = Matrix(ellipse)
matrix.draw_ellipses(ellipse)
matrix.display()
matrix.count_regions()
model_x  =  20; half_range_x  = 1; step_x  = 1
model_y  =  46; half_range_y  = 1; step_y  = 1
model_rx = 166; half_range_rx = 1; step_rx = 1
model_ry = 264; half_range_ry = 1; step_ry = 1
for    x  in range(-half_range_x , half_range_x  + 1, step_x
):
for   y  in range(-half_range_y , half_range_y  + 1, step_y
):
  for  rx in range(-half_range_rx, half_range_rx + 1,
step_rx):
   for ry in range(-half_range_ry, half_range_ry + 1,
step_ry):
    ellipse = Ellipse(model_x + x,model_y + y, model_rx +
rx,model_ry + ry)
    matrix  = Matrix(ellipse)
    matrix.draw_ellipses(ellipse)
    matrix.count_regions()
# "
```

Створивши програмний модуль 6.1., ми отримали діаграму взаємодії п’ятьох членів команди розробників програмних систем з використанням хмарної технології за фаховим рівнем та врахуванням психотипів її членів (рис. 6.8).

Рисунок 6.8 – Модель формування команд розробників програмних систем за фаховим рівнем та з врахуванням психотипів:
A, B, C, D, F – член команди розробників програмних систем;
8, 6, 2, 5, 3 – належність до певного психотипу

Де, великими латинськими буквами відображено графічне представлення певного члена команди розробників програмних систем, відповідним кольором фаховий рівень компетенції, а цифрою його приналежність до певного психотипу.

Під час формування команд традиційними методами усі площини будуть білими, оскільки нам не відомий психотип того чи іншого члена команди, і він стане проявлятися лише згодом, у процесі виконання певних поставлених завдань із розроблення програмних Web-систем з використанням хмарної технології. Тому велика ймовірність того, що у команду розробників може потрапити більше ніж один член із керівними психотипами, що негативно буде відображатися на внутрішньому мікрокліматі в команді. Оскільки два члени команд із

керівними психотипами і більше будуть намагатися домінувати у команді розробників, що водночас може створити два і більше центри прийняття рішень, а це негативно впливатиме на кінцевий результат виконання поставлених програмних завдань (рис. 6.9).

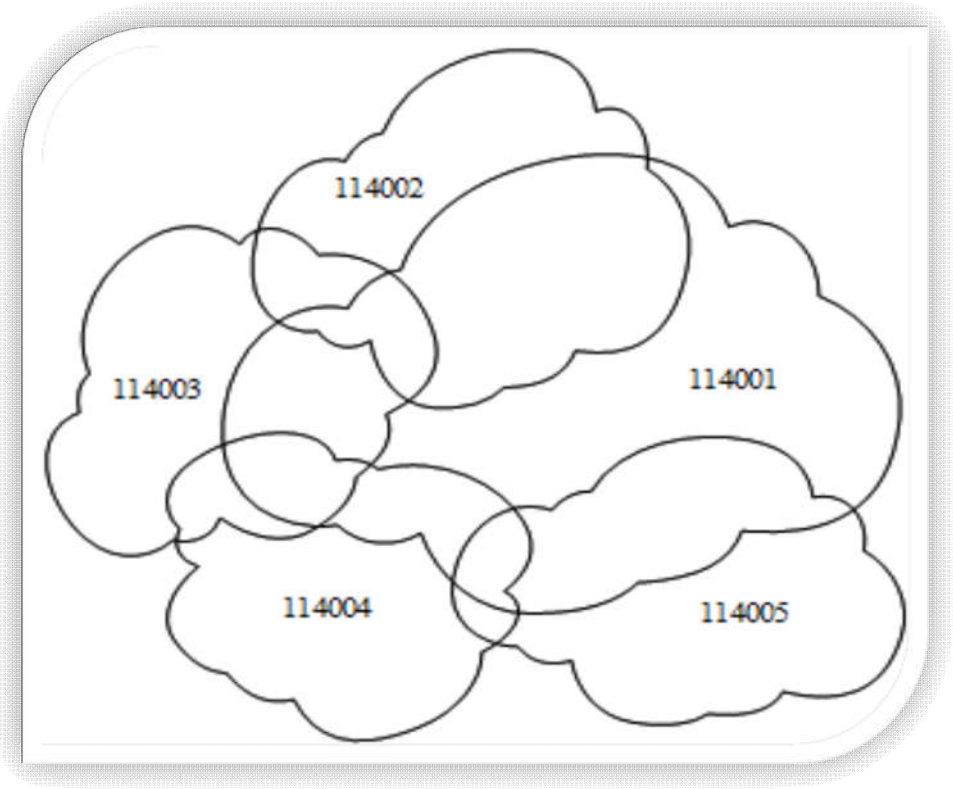


Рисунок 6.9 – Модель формування команди розробників програмних систем традиційними підходами

На практиці такі випадки трапляються досить часто. Коли є керівник групи розробників програмних систем з використанням хмарної технології з опрацювання інформаційних потоків даних на незалежних обчислювальних платформах, водночас є неформальний лідер (керівник), до якого прислухається уся команда, і така ситуація вкрай не бажана. Тому для розв’язання цієї проблеми і було запропоновано використовувати інформацію про психотипи кожного із претендентів у команди розробників ще до початку розробки програмної системи, унеможливлене потрапляння двох членів команд із керівними характеристиками психотипів та більше.

6.4. Емпіричні методи досліджень та математичне опрацювання експериментальних даних

Для проведення експерименту використано емпіричні методи дослідження, які би підтвердили або спростували гіпотезу. Ця гіпотеза полягає у тому, що при формуванні команд розробників програмних Web-систем з використанням хмарної технології необхідно враховувати психотип кожного члена команди при долучені останнього до команди, що може дати позитивний результат. А саме формування команд розробників програмних Web-систем з використанням хмарної технології із врахуванням психотипів їх майбутніх членів мало б дати позитивний кінцевий результат у кількісному еквіваленті успішно закінчених проєктів. Для проведення експерименту та для перевірки гіпотези сформуємо 24 команди розробників програмних систем, 12 із яких – за методами групової динаміки та з врахуванням психотипів членів команд, а 12 – без врахування психотипів її членів.

Отже, для формування команд розробників програмних Web-систем методами групової динаміки з визначенням та врахуванням психотипів кожного із їхніх членів необхідно заздалегідь визначити психотипи претендентів у команди з розроблення програмних систем. Для цього проведемо тестування претендентів у члени команд розробників із визначення їх психотипів згідно з розробленою програмою, яка складається з 81 запитання. Результати тестування та формування команд із врахуванням психотипів розробників програмних систем наведено у табл. 6.1.

Кожному члену команд розробників програмних систем надано код для анонімної оцінки результатів діяльності тієї чи іншої команди. Тоді склад та належність 24-х команд розробників буде такий: по 5 членів у кожній команді із двох закладів вищої освіти, а саме з ДВНЗ «Прикарпатський національний університет імені Василя Стефаника» та Івано-Франківського національного технічного університету нафти і газу.

Отже, загальна кількість членів команд, які брали участь в експерименті для підтвердження або спростування гіпотези, становить 120 осіб, 82 із них – чоловіки та 38 – жінки. Для правильного проведення експерименту було визначено, що команди будуть формуватися по 12 у кожному закладів вищої освіти, 6 із яких формуються за методами групової динаміки з врахуванням психотипів членів команд

розробників програмних Web-систем у такий спосіб, щоб у команду потрапляв лише один претендент із керівним психотипом, а 6 – команд формувалися у звичайний спосіб.

Таблиця 6.1 – Формування команд розробників із врахуванням їхніх психотипів

Шифр члена команди розробників	Реформатор	Поміч-ник	Моти-ватор	Роман-тик	Мисли-тель	Лояліст	Ентузі-аст	Лідер	Посере-дник
	Психотип розробника програмних систем								
	1	2	3	4	5	6	7	8	9
101001								□	
101002						□			
101003		□							
101004					□				
101005			□						
102001	□								
102002							□		
102003									□
102004						□			
102005			□						
103001								□	
103002	□								
103003				□					
103004							□		
103005		□							
104001						□			
104002									□
104003								□	
104004				□					
104005					□				
105001			□						
105002							□		
105003									
105004								□	
105005									□
106001	□								
106002			□						

106003									□
106004						□			
106005					□				
107001			□						
107002	□								
107003									□
107004						□			
107005				□					
108001							□		
108002			□						□
108003		□						□	
108004					□				
108005			□						
109001								□	
109002	□								
109003			□						
109004						□			
109005									□
110001					□				
110002	□								
110003						□			
110004								□	
110005									□
111001							□		
111002			□						
111003					□				
111004				□					
111005						□			
112001				□					
112002		□							
112003									□
112004						□			
112005			□						

Також у кожному із закладів вищої освіти було сформовано по одній команді лише з чоловіків та жінок, щоб можна було перевірити ще одну нашу гіпотезу, що змішані команди значно краще взаємоді-

ють між собою, ніж моногамні. Результати формування команд розробників програмних систем наведено у табл. 6.2.

Таблиця 6.2 – Формування команд для експерименту

№ п/п	№ команди та належність	Формування	Гендер	К-сть членів
1	**2**	**3**	**4**	**5**
1	Команда № 1П	Групова динаміка	Змішана	5
2	Команда № 2П	Групова динаміка	Змішана	5
3	Команда № 3П	Класичне	Жіноча	5
4	Команда № 4П	Групова динаміка	Чоловіча	5
5	Команда № 5П	Класичне	Змішана	5
6	Команда № 6П	Групова динаміка	Змішана	5
7	Команда № 7П	Класичне	Змішана	5
8	Команда № 9П	Класичне	Чоловіча	5
9	Команда № 10 П	Групова динаміка	Змішана	5
10	Команда № 11П	Класичне	Змішана	5
11	Команда № 12П	Групова динаміка	Жіноча	5
12	Команда № 13П	Класичне	Змішана	5
13	Команда № 1Н	Групова динаміка	Змішана	5
14	Команда № 2Н	Класичне	Змішана	5
15	Команда № 3Н	Групова динаміка	Змішана	5
16	Команда № 4Н	Класичне	Змішана	5
17	Команда № 5Н	Групова динаміка	Жіноча	5
18	Команда № 6Н	Класичне	Жіноча	5
19	Команда № 7Н	Групова динаміка	Змішана	5
20	Команда № 8Н	Класичне	Змішана	5
21	Команда № 9Н	Групова динаміка	Змішана	5

Продовження табл. 6.2

1	2	3	4	5
22	Команда № 10Н	Класичне	Змішана	5
23	Команда № 11Н	Групова динаміка	Чоловіча	5
24	Команда № 12Н	Класичне	Чоловіча	5

Наступним етапом проведення експерименту було формування 24 типових завдань, які випадково були розіграні між командами розробників програмних систем з використанням хмарної технології. Також зазначимо, що керівника групи розробників програмної Web-системи обирали самостійно лише ті члени групи, які формувалися довільно, а у командах, де враховувалися психотипи, керівник призначався із відповідним психотипом керівника. Приклади типових завдань наведено у Додатку Б.

Для правильного проведення експерименту усі команди мали практично однакові програмно-технічні комплекси із відповідним програмним забезпеченням, а також були обмеженні у часі на розроблення програмної Web-системи з використанням хмарної технології для опрацювання інформаційних потоків даних на незалежних обчислювальних платформах. Розробники мали 240 годин, тобто десять днів, завдяки чому уникнули шахрайства щодо часу виконання завдання.

Після проведення експерименту з підтвердження або спростування гіпотези комісія оцінювала розроблені програмні Web-системи з використанням хмарної технології за такими критеріями:

- успішне завершення розроблення програмної Web-системи (від 90% до 100%) у певний термін із дотриманням усіх поставлених вимог;
- часткове розроблення програмної Web-системи (більше 60% до 90%) у встановлений термін із частковим дотриманням усіх поставлених вимог до завдання;
- часткове розроблення програмної Web-системи (менше 60%) у встановлений термін із частковим дотриманням усіх поставлених вимог до завдання.

Усі отримані результати проведення емпіричного експерименту наведено у зведені табл. 6.3. для проведення математичного аналізу.

Таблиця 6.3 – Результати виконання експериментальних завдань проєктними командами

№	Перелік завдань для проведення експеременту	Команди розробників програмних систем																							
		Івано-Франківський національний технічний університет нафти і газу												ДВНЗ «Прикарпатський національний університет імені Василя Стефаника»											
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
1	Побудуйте структурну схему проєкту та взаємозв'язки між модулями	100	100	70	80	60	80	100	70	100	60	100	60	70	100	100	100	60	70	80	60	90	60	70	60
2	Спроєктуйте структуру бази даних та взаємозв'язків	100	100	60	70	50	80	100	60	100	60	80	70	50	100	90	90	60	70	80	60	90	60	70	60
3	Спроєктуйте інтерфейс користувача згідно з вимогами ISO 9001	100	95	50	70	50	70	100	60	90	50	90	60	60	90	100	80	40	60	70	70	80	50	80	50
4	Сконструюйте режим демо та повноцінної програми	100	90	40	60	60	70	100	70	80	50	90	70	50	90	100	90	50	70	70	70	80	60	70	50
5	Забезпечте шифрування даних, які зберігаються в базі	100	100	50	60	50	50	90	80	90	40	80	80	70	80	90	90	50	70	80	70	70	70	80	40
6	Забезпечте ідентифікацію або авторизацію входу у вашу програму	100	90	90	60	60	80	100	60	100	30	80	60	60	80	90	100	60	60	70	60	80	60	70	50
7	Передбачте автоматичне відновлення даних у вашій програмі	80	100	70	70	60	70	90	60	90	40	80	60	60	90	100	80	70	60	60	70	70	50	60	40
8	Побудуйте інфологічну модель, визначте обмеження на атрибути та сутності, визначте зв`язки між сутностями. При виборі сутностей обгрунтуйте їхню необхідність. Сформулюйте всі обмеження природною мовою. При визначенні зв`язків необхідно обгрунтовувати вид зв`язку, його потужність, обов`язковість	80	90	60	70	60	70	90	60	80	50	90	70	70	100	90	100	80	60	70	80	80	70	70	50
9	Здійсніть декомпозицію загальної задачі та визначте групи користувачів у вашій системі та виконувані ними функції	100	100	60	60	50	80	90	50	100	60	80	70	70	100	90	90	50	60	80	70	70	60	60	50
10	Визначте кількість і види додатків, які потрібно розробити для реалізації загальної поставленої задачі у вашій системі. Задайте функціональне призначення кожного додатка	90	100	50	60	50	70	90	70	100	50	90	80	60	90	100	90	60	80	70	70	80	70	70	60
11	Виберіть найбільш характерні функції для більшості додатків і задайте їх як збережені процедури. Задайте словесний алгоритм виконання збережених процедур	100	100	60	70	50	80	100	60	90	50	80	80	70	80	90	90	60	70	90	60	80	70	60	60
12	Визначте необхідні тригери в системі. Задайте їхнє призначення та функціональні дії, що вимагаються	80	100	40	60	50	80	90	70	100	60	90	60	70	100	100	90	50	70	70	70	90	80	60	60
13	Для одного вибраного додатка здійсніть проєктування, визначивши множину станів і їхнє відображення в термінах інтерфейсу користувача та множину переходів із зазначенням сигналів впливів, які викликають ці переходи. Задайте в умовних позначеннях зовнішній вигляд екранних форм для цього додатка	80	100	50	60	50	80	100	70	80	50	90	70	60	100	90	90	60	70	80	70	80	70	60	50
14	Проведіть критичне тестування вашої розробки програмної системи	80	90	40	60	40	80	90	70	90	60	100	80	60	100	100	100	50	70	90	80	80	80	70	50
Відсоток виконання поставлених завдань		**92**	**97**	**56**	**65**	**53**	**74**	**95**	**65**	**92**	**51**	**87**	**69**	**63**	**93**	**95**	**91**	**57**	**67**	**76**	**69**	**80**	**65**	**68**	**52**

Ґрунтуючись на отриманих даних (табл. 6.3), можемо провести ряд математичних досліджень, а саме визначити відсоток програмних завдань із розроблення Web-систем з використанням хмарної технології, які були завершені успішно, частково успішно та не успішно. Цей аналіз дає нам змогу перевірити коректність проведення експерименту, оскільки ми знаємо розподіл відсотків за успішністю розроблення програмних Web-систем, а саме: успішні – 15 – 35%, частково успішні – 35 – 55% та неуспішні – 15 – 35%.

Проведемо детальний аналіз успішності розроблення програмних Web-систем згідно з поставленими експериментальними завданнями за ДВНЗ «Прикарпатському національному університеті імені Василя Стефаника», успішно завершено чотири проєкти з розробки програм-

них Web-систем з використанням хмарної технології відповідно до вимог, у п'яти були частково дотримані вимоги більше ніж на 60%, а три – не успішні, оскільки не вклалися в часові обмеження та виконали поставлені вимоги менше ніж на 60% (рис 6.10).

Рисунок 6.10 – Аналіз виконання поставлених завдань у ДВНЗ «Прикарпатський національний університет ім. Василя Стефаника»

У наступному кроці проаналізуємо успішність розроблення програмних Web-систем з використанням хмарної технології згідно з поставленими завданнями в Івано-Франківському національному технічному університеті нафти і газу, де успішно завершено три проєкти відповідно до вимог, у семи – частково були дотримані вимоги, понад 60%, а два виявилися не успішними, оскільки закінчились із дотриманням менше 60% вимог (рис. 6.11).

Провівши емпіричний аналіз успішності розроблення програмних Web-систем та виконавши математичне узагальнення, ми сформували узагальнену діаграму успішності виконання поставлених завдань згідно з критеріями, які наведені вище, за двома закладами вищої освіти.

Сумарна успішність за двома закладами вищої освіти становить: 7 проєктів із розроблення програмних систем закінчились успішно (29,20%), 12 проєктів закінчились частково успішно (50,00%) і 5 проєктів були не успішними (20,80%). Діаграму результатів наведено на рис. 6.12.

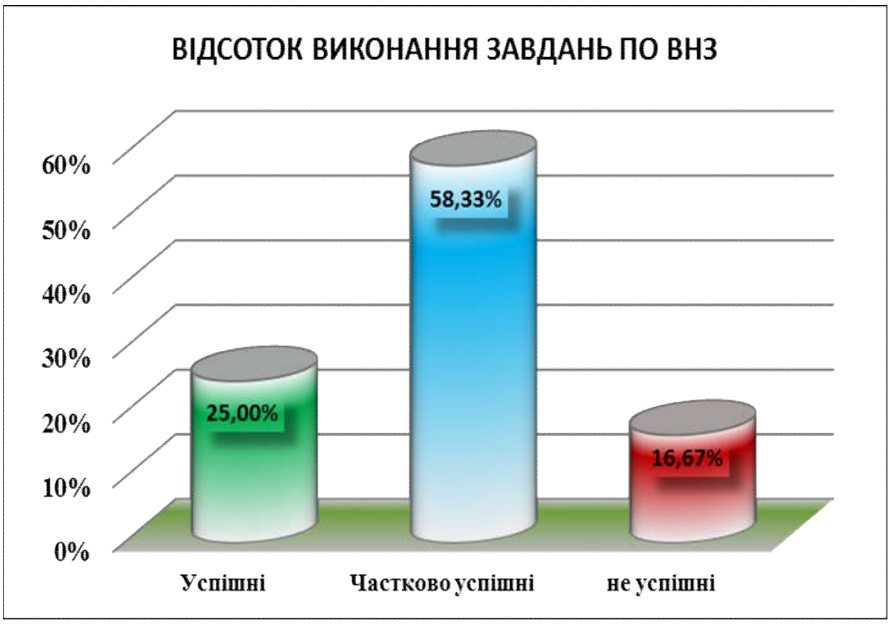


Рисунок 6.11 – Аналіз виконання поставлених завдань в Івано-Франківському національному технічному університеті нафти і газу

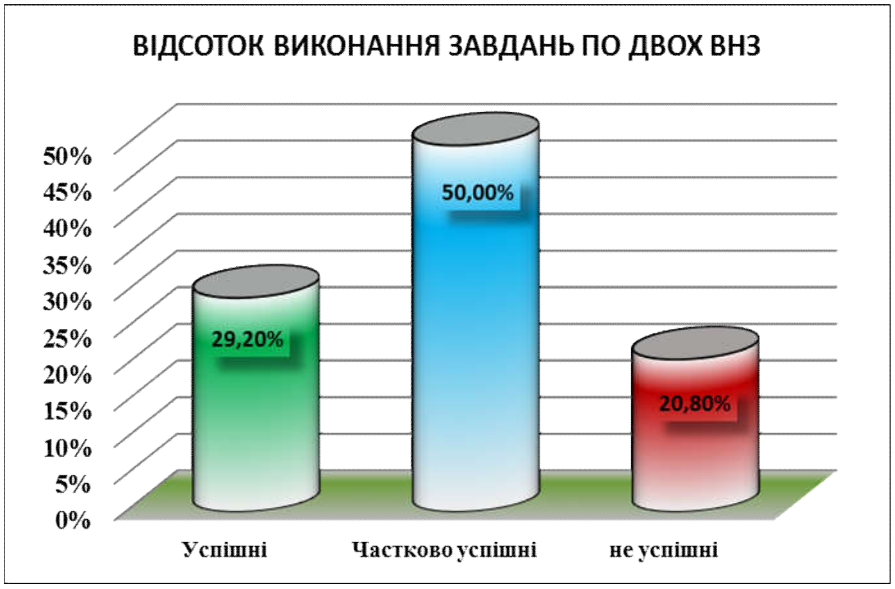


Рисунок 6.12 – Аналіз виконання поставлених завдань по двох ВНЗ

Для візуалізації інформації за виконаними поставленими завданнями сформуємо зведену діаграму, де на графіку відображено ефективність виконання поставлених завдань з огляду на три критерії, які успішно завершились, частково та не успішні у розрізі команд закладами вищої освіти та брали участь в експерименті із підтвердження або спростування гіпотези (рис. 6.13).

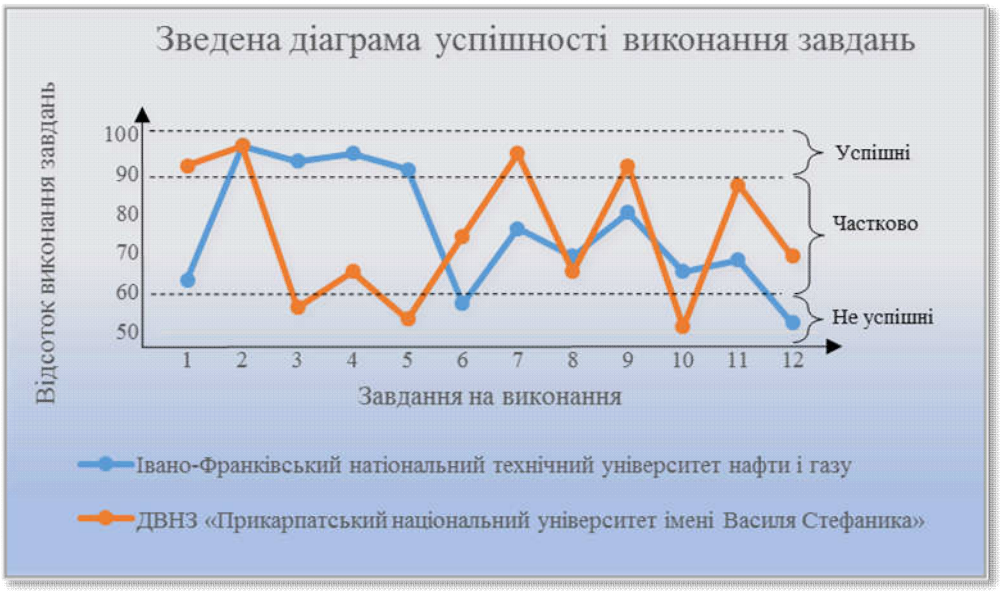


Рисунок 6.13 – Діаграма успішності виконання поставлених завдань за закладами вищої освіти

Проведемо усереднені математичні розрахунки за усіма 24-х командами розробників програмних Web-систем з використанням хмарної технології, які частково формувалися з застосуванням інформації про психотипи їхніх членів, а також частково таких команд, які формувалися за традиційними підходами та засадами, і виконували поставлені завдання для перевірки гіпотези емпіричним методом. Згрупуємо команди за типами підходів у їхньому формуванні у відповідні кластери для проведення порівняльного кластерного аналізу (табл. 6.4).

Таблиця 6.4 – Групування команд програмних Web-систем за типом формування

№ п/п	№ команди та належність	Формування	Гендер	К-сть членів	Сума балів
1	**2**	**3**	**4**	**5**	**6**
1	Команда № 1П	Групова динаміка	Мішана	5	92
2	Команда №2 П	Групова динаміка	Мішана	5	97
4	Команда № 4П	Групова динаміка	Мішана	5	65
6	Команда № 6П	Групова динаміка	Чоловіча	5	74
9	Команда № 10П	Групова динаміка	Мішана	5	92
11	Команда № 12П	Групова динаміка	Жіноча	5	87
13	Команда № 1Н	Групова динаміка	Мішана	5	63
15	Команда № 3Н	Групова динаміка	Мішана	5	95
17	Команда № 5Н	Групова динаміка	Жіноча	5	57

Продовження табл. 6.4

1	2	3	4	5	6
19	Команда № 7Н	Групова динаміка	Мішана	5	75
21	Команда № 9Н	Групова динаміка	Мішана	5	80
23	Команда № 11Н	Групова динаміка	Чоловіча	5	68
3	Команда № 3П	Класичне	Жіноча	5	56
5	Команда № 5П	Класичне	Мішана	5	53
7	Команда № 7П	Класичне	Мішана	5	95
8	Команда № 9П	Класичне	Чоловіча	5	65
10	Команда № 11П	Класичне	Мішана	5	51
12	Команда № 13П	Класичне	Мішана	5	69
14	Команда № 2Н	Класичне	Мішана	5	93
16	Команда № 4Н	Класичне	Мішана	5	91
18	Команда № 6Н	Класичне	Жіноча	5	67
20	Команда № 8Н	Класичне	Мішана	5	69
22	Команда № 10Н	Класичне	Мішана	5	65
24	Команда № 12Н	Класичне	Чоловіча	5	52

На основі трансформованої зведеної табл. 6.4 сформуємо табл. 6.5, яка відображає отриману інформацію про середньозважений відсоток виконання поставлених під задач, як експериментальними групами, так і еталонними. Експериментальні групи розробників програмних систем з використанням хмарної технології – це такі групи, які сформовані з огляду на психотипи її членів, а еталонні – це ті, сформовані традиційними методами. Результат обчислимо за математичним виразом (6.1):

$$\Delta = (\sum_{i=1}^{n} ЕГ_i - \sum_{i=1}^{n} КГ_i)/n = \frac{1107{,}06 - 964{,}17}{14} = 10{,}21\%, \quad (6.1)$$

де Δ – різниця відсотків виконання завдань між двома кластерами; n – кількість під завдань для команд певної групи кластера, які брали участь в експерименті; $ЕГ_{ij}$ – експериментальні групи кластера; $КГ_{ij}$ – контрольні групи кластера.

Ефективність команд розробників програмних Web–систем з використанням Хмарної технології за двома сформованими кластерами (з огляду на психотип та без нього) наведено на рис 6.14.

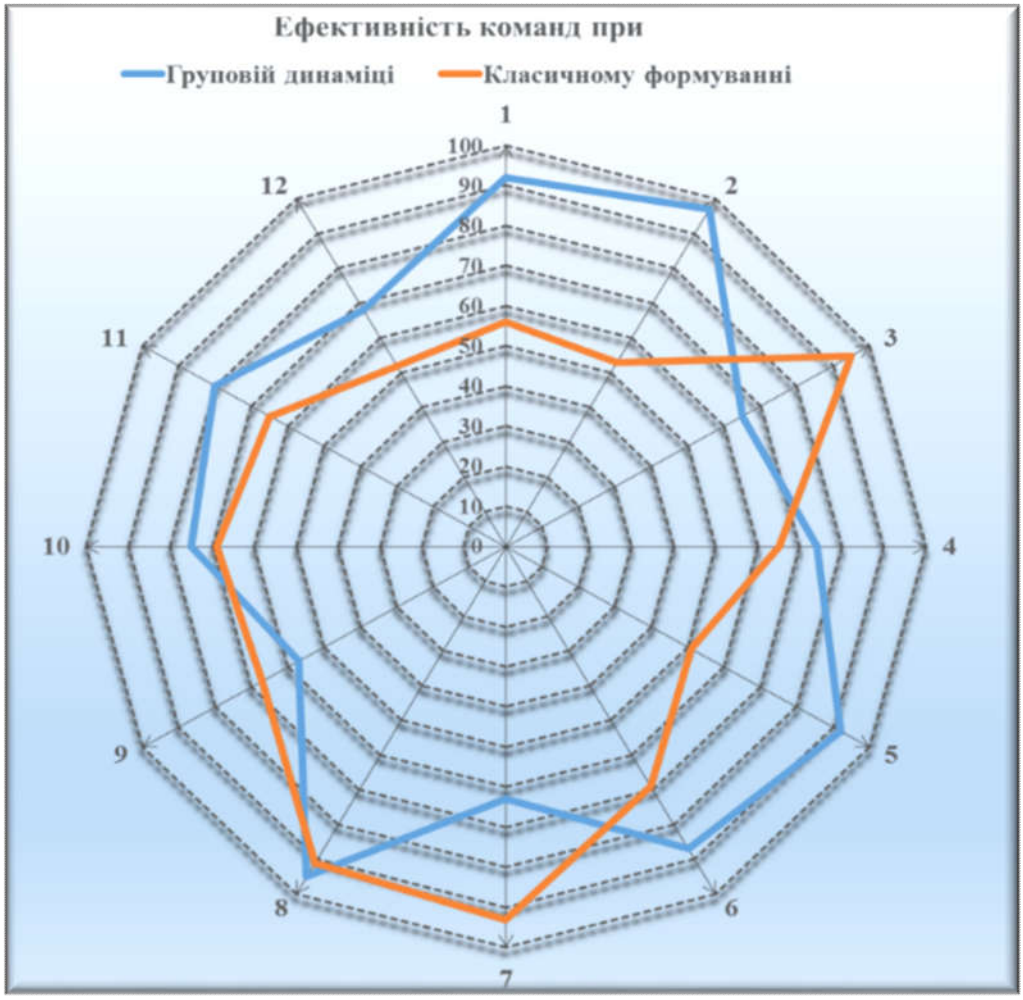


Рисунок 6.14 – Кластерний аналіз розроблення програмних систем за різним принципом формування команд

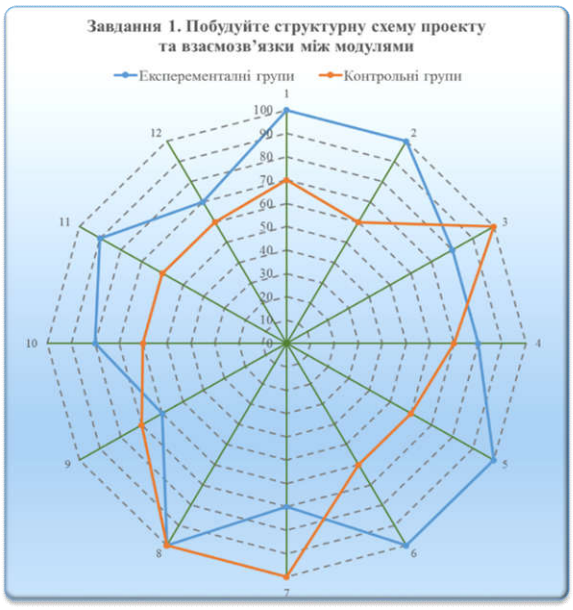
Провівши кластерний аналіз виконання завдань у командах розробників програмних систем у розрізі розв'язання підзадач як експериментальними групами, так і командами, які були сформовані за традиційним підходом (контрольні групи). Визначимо ефективність запропонованої моделі формування команд розробників програмних систем з огляду на психотипи їхніх членів згідно з експериментальними даними (табл. 6.5), що становило приріст на 10,21% при реалізації поставлених завдань. А також різницю між результатами кластерного аналізу, які сформовані з експериментальних та контрольних груп за кожного із підзадач окремо відповідно математичного виразу (6.2):

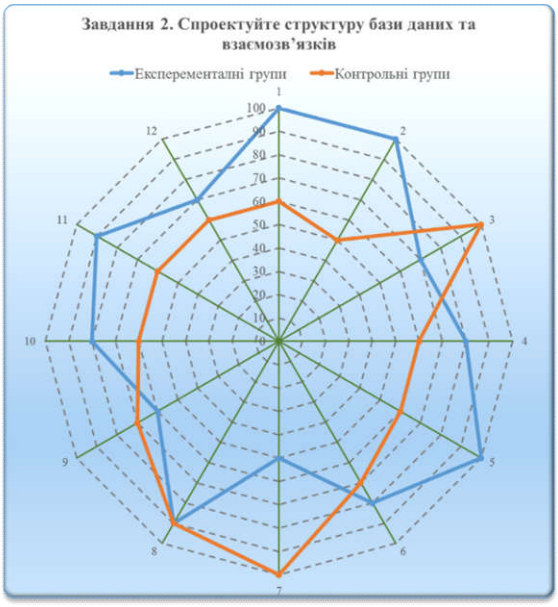
$$\Delta_j = \sum_{i=1}^{n} \text{ЕГ}_{ij} - \sum_{i=1}^{n} \text{КГ}_{ij}, \tag{6.2}$$

де Δj – різниця відсотків виконання j-го підзавдання з розроблення програмної системи; n – кількість команд певної групи, які брали участь в експерименті; $ЕГ_{ij}$ – експериментальні групи; $КГ_{ij}$ – контрольні групи.

Таблиця 6.5 – Виконання завдань командами розробників програмних систем

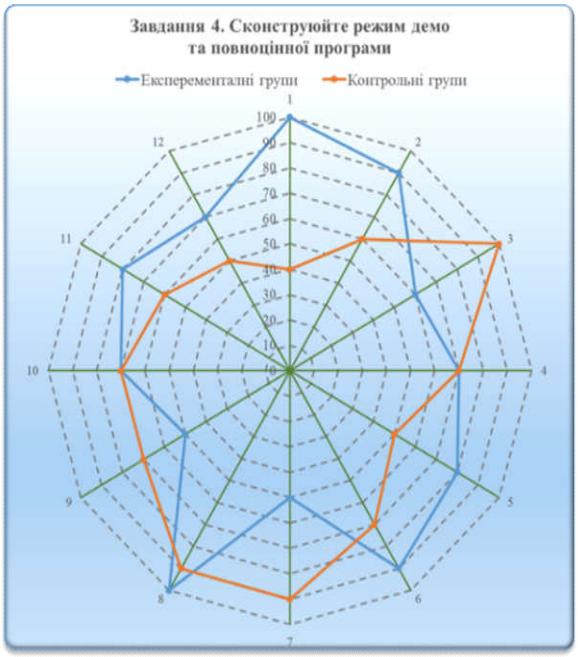
№	Перелік завдань для проведення експерименту		Команди розробників програмних систем												
		ЕГ	1	2	4	6	9	11	13	15	17	19	21	23	Відсоток
		КГ	3	5	7	8	10	12	14	16	18	20	22	24	виконання
1	Побудуйте структурну схему проєкту та взаємозв'язки між модулями	ЕГ	100	100	80	80	100	100	70	100	60	80	90	70	85,83
		КГ	70	60	100	70	60	60	100	100	70	60	60	60	72,50
2	Спроєктуйте структуру бази даних та взаємозв'язків	ЕГ	100	100	70	80	100	80	50	90	60	80	90	70	80,83
		КГ	60	50	100	60	60	70	100	90	70	60	60	60	70,00
3	Спроєктуйте інтерфейс користувача згідно з вимогами ISO 9001	ЕГ	100	95	70	70	90	90	60	100	40	70	80	80	78,75
		КГ	50	50	100	60	50	60	90	80	60	70	50	50	64,17
4	Сконструюйте режим демо та повноцінної програми	ЕГ	100	90	60	70	80	90	50	100	50	70	80	70	75,83
		КГ	40	60	100	70	50	70	90	90	70	70	60	50	68,33
5	Забезпечте шифрування даних, які зберігаються в базі	ЕГ	100	100	60	50	90	80	70	90	50	80	70	80	76,67
		КГ	50	50	90	80	40	80	80	90	70	70	70	40	67,50
6	Забезпечте ідентифікацію або авторизацію входу у вашу програму	ЕГ	100	90	60	80	100	80	60	90	60	70	80	70	78,33
		КГ	90	60	100	60	30	60	80	100	60	60	60	50	67,50
7	Передбачте автоматичне відновлення даних у вашій програмі	ЕГ	80	100	70	70	90	80	60	100	70	60	70	60	75,83
		КГ	70	60	90	60	40	60	90	80	60	70	50	40	64,17
8	Побудуйте інфологічну модель, визначте обмеження на атрибути та сутності, визначте зв`язки між сутностями. При виборі сутностей обгрунтуйте їхню необхідність. Сформулюйте всі обмеження природною мовою. При визначенні зв`язків необхідно обгрунтовувати вид зв`язку, його потужність, обов`язковість	ЕГ	80	90	70	70	80	90	70	90	80	70	80	70	78,33
		КГ	60	60	90	60	50	70	100	100	60	80	70	50	70,83
9	Здійсніть декомпозицію загальної задачі та визначте групи користувачів у вашій системі та виконувані ними функції	ЕГ	100	100	60	80	100	80	70	90	50	80	70	60	78,33
		КГ	60	50	90	50	60	70	100	90	60	70	60	50	67,50
10	Визначте кількість і види додатків, які потрібно розробити для реалізації загальної поставленої задачі у вашій системі. Задайте функціональне призначення кожного додатка	ЕГ	90	100	60	70	100	90	60	100	60	70	80	70	79,17
		КГ	50	50	90	70	50	80	90	90	80	70	70	60	70,83
11	Виберіть найбільш характерні функції для більшості додатків і задайте їх як збережені процедури. Задайте словесний алгоритм виконання збережених процедур	ЕГ	100	100	70	80	90	80	70	90	60	90	80	60	80,83
		КГ	60	50	100	60	50	80	80	90	70	60	70	60	69,17
12	Визначте необхідні тригери в системі. Задайте їхнє призначення та функціональні дії, що вимагаються	ЕГ	80	100	60	80	100	90	70	100	50	70	90	60	79,17
		КГ	40	50	90	70	60	60	100	90	70	70	80	60	70,00
13	Для одного вибраного додатка здійсніть проєктування, визначивши множину станів і їхнє відображення в термінах інтерфейсу користувача та множину переходів із зазначенням сигналів впливів, які викликають ці переходи. Задайте в умовних позначеннях зовнішній вигляд екранних форм для цього додатка	ЕГ	80	100	60	80	80	90	60	90	60	80	80	60	76,67
		КГ	50	50	100	70	50	70	100	90	70	70	70	50	70,00
14	Проведіть критичне тестування вашої розробки програмної системи	ЕГ	80	90	60	80	90	100	60	100	50	90	80	70	79,17
		КГ	40	40	90	70	60	80	100	100	70	80	80	50	71,67

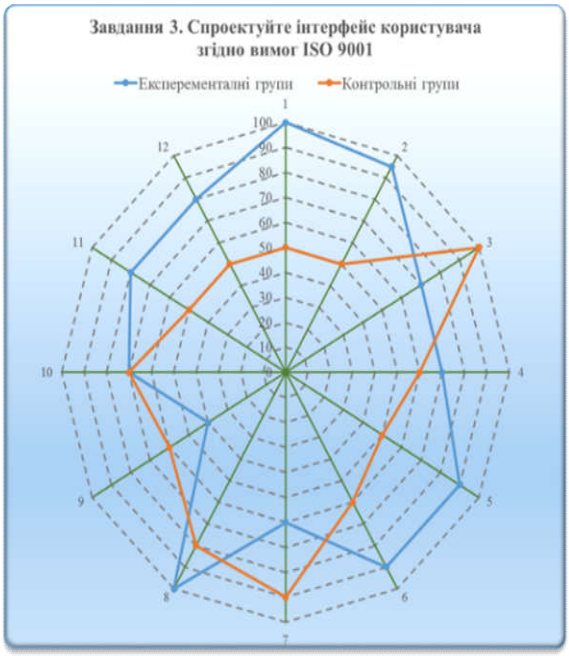




$$\Delta_1 = \sum_{i=1}^{n} ЕГ_{i1} - \sum_{i=1}^{n} КГ_{i1} = 13{,}33\%;$$

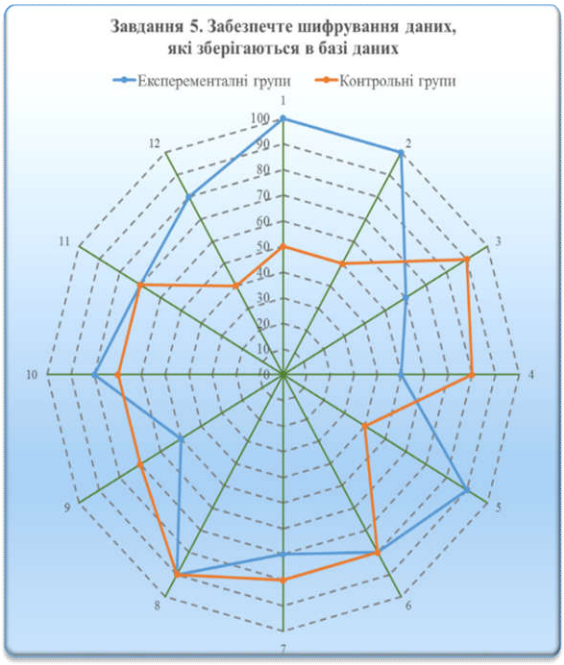
$$\Delta_2 = \sum_{i=1}^{n} ЕГ_{i2} - \sum_{i=1}^{n} КГ_{i2} = 10{,}83\%$$

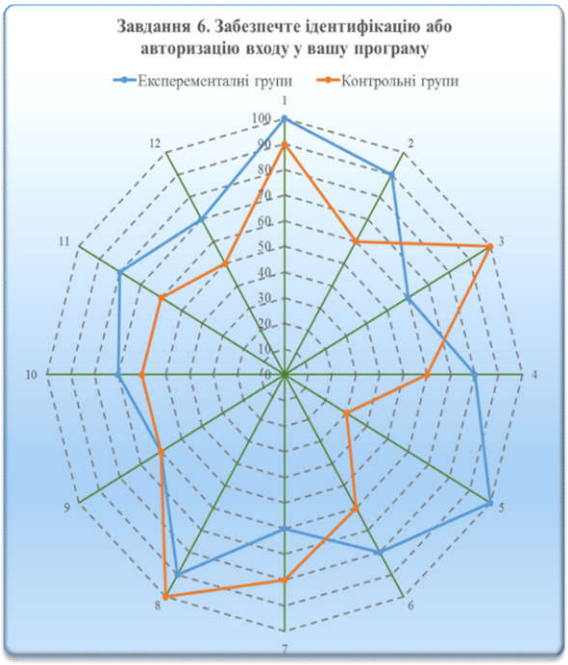




$$\Delta_3 = \sum_{i=1}^{n} ЕГ_{i3} - \sum_{i=1}^{n} КГ_{i3} = 14{,}58\%;$$

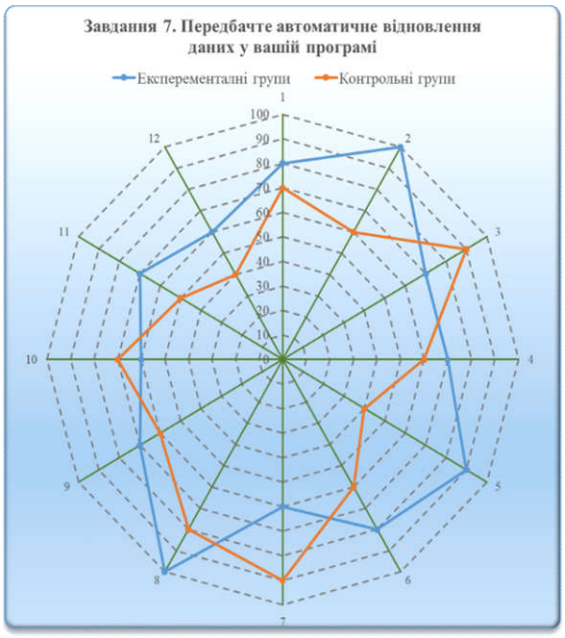
$$\Delta_4 = \sum_{i=1}^{n} ЕГ_{i4} - \sum_{i=1}^{n} КГ_{i4} = 7{,}50\%$$

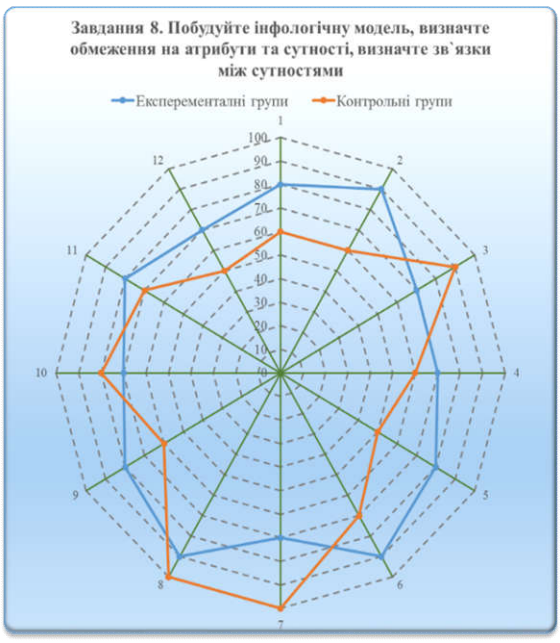




$$\Delta_5 = \sum_{i=1}^{n} ЕГ_{i5} - \sum_{i=1}^{n} КГ_{i5} = 9{,}17\%;$$

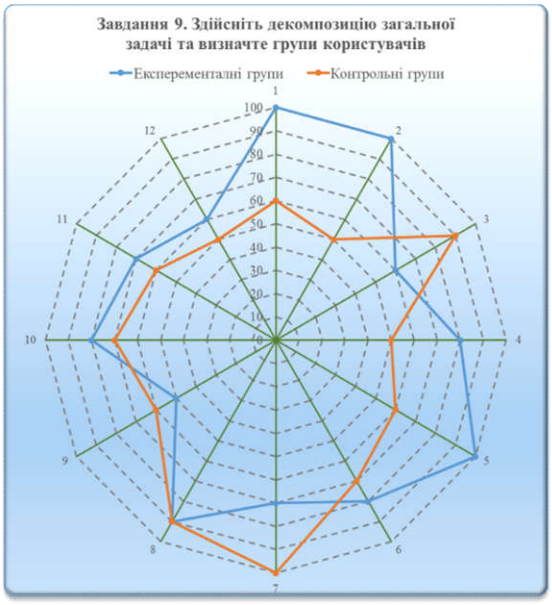
$$\Delta_6 = \sum_{i=1}^{n} ЕГ_{i6} - \sum_{i=1}^{n} КГ_{i6} = 10{,}83\%$$

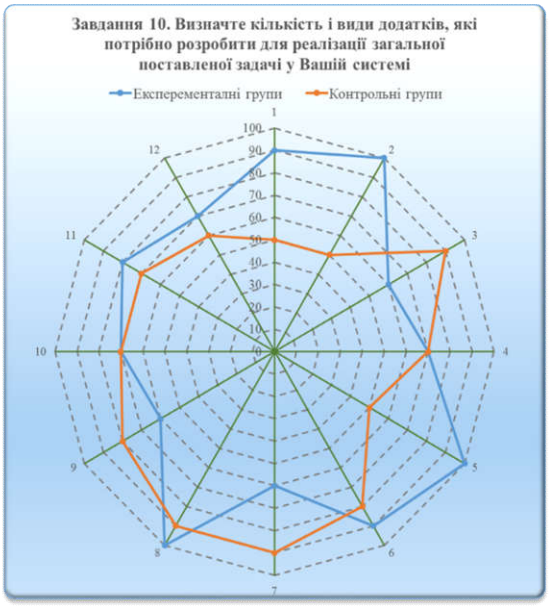




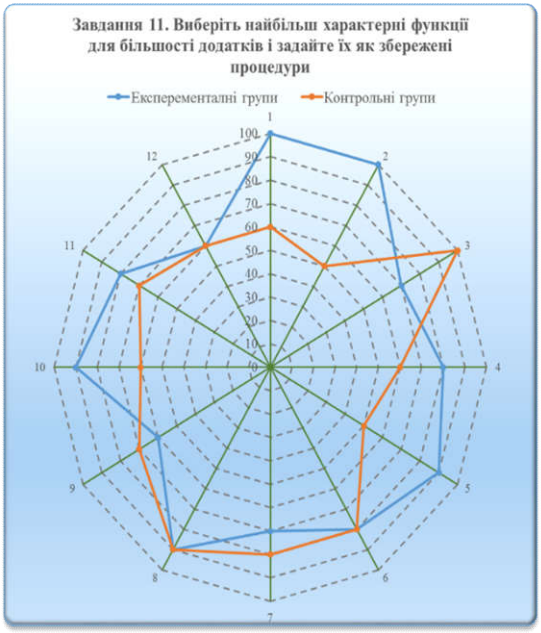
$$\Delta_7 = \sum_{i=1}^{n} ЕГ_{i7} - \sum_{i=1}^{n} КГ_{i7} = 11{,}67\%;$$

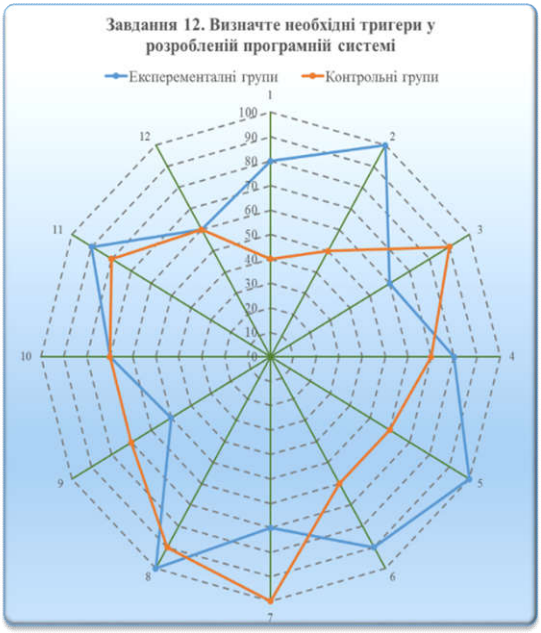
$$\Delta_8 = \sum_{i=1}^{n} ЕГ_{i8} - \sum_{i=1}^{n} КГ_{i8} = 7{,}50\%$$



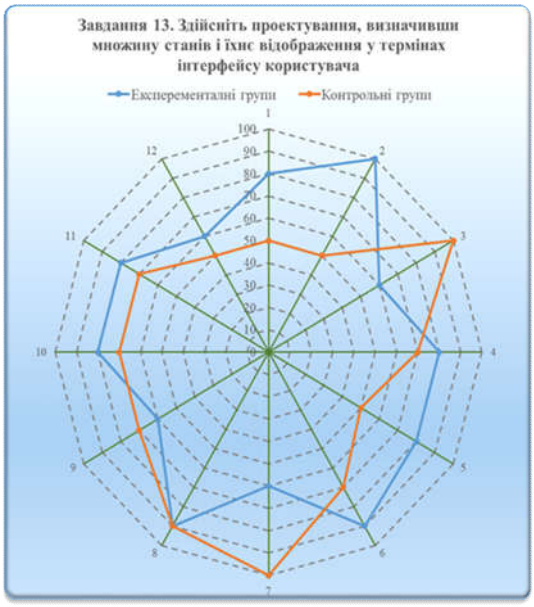


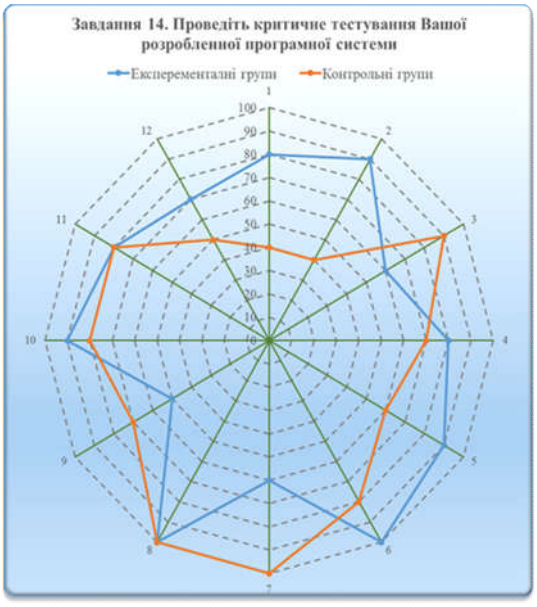
$$\Delta_9 = \sum_{i=1}^{n} ЕГ_{i9} - \sum_{i=1}^{n} КГ_{i9} = 10{,}83\%; \quad \Delta_{10} = \sum_{i=1}^{n} ЕГ_{i10} - \sum_{i=1}^{n} КГ_{i10} = 8{,}3\%$$





$$\Delta_{11} = \sum_{i=1}^{n} ЕГ_{i11} - \sum_{i=1}^{n} КГ_{i11} = 11{,}6\%; \quad \Delta_{12} = \sum_{i=1}^{n} ЕГ_{i12} - \sum_{i=1}^{n} КГ_{i12} = 9{,}2\%$$





$$\Delta_{13} = \sum_{i=1}^{n} ЕГ_{i13} - \sum_{i=1}^{n} КГ_{i13} = 6{,}7\%; \quad \Delta_{14} = \sum_{i=1}^{n} ЕГ_{i14} - \sum_{i=1}^{n} КГ_{i14} = 7{,}5\%$$

Для цього визначимо відсоток виконання кожної із підзадач як експериментальними групами, так і контрольними, а результати наведемо в табл. 6.6.

Ґрунтуючись на експериментальних даних, проведемо детальний аналіз виконання поставлених завдань із розроблення програмних систем за способом формування команд, а саме із врахуванням психотипів її членів або традиційним способом за двома університетами, де зафіксуємо усереднені значення за трьома критеріями якості виконання завдань згідно з математичним виразом (6.3):

$$Q_{и} = \frac{\sum Q_{и1} + \sum Q_{и2}}{2} * 100, \tag{6.3}$$

де $\sum Q_{и1}$ – сума успішних та частково успішних виконаних проєктів (понад 60%), які відповідали вимогам замовника у ДВНЗ «Прикарпатський національний університет імені Василя Стефаника»; $\sum Q_{и2}$– сума успішних та частково успішних виконаних проектів (понад 60%), які відповідали вимогам замовника в Івано-Франківському національному університеті нафти і газу (рис. 6.15).

Таблиця 6.6 – Кластерний аналіз виконання підзавдань групами розробників

№	Перелік завдань для проведення експерименту	Тип груп	%	Δ=Σег-Σкг
1	Побудуйте структурну схему проєкту та взаємозв'язки між модулями	ЕГ	85,83	13,33
		КГ	72,50	
2	Спроєктуйте структуру бази даних та взаємозв'язків	ЕГ	80,83	10,83
		КГ	70,00	
3	Спроєктуйте інтерфейс користувача згідно з вимогами ISO 9001	ЕГ	78,75	14,58
		КГ	64,17	
4	Сконструюйте режим демо та повноцінної програми	ЕГ	75,83	7,50
		КГ	68,33	
5	Забезпечте шифрування даних, які зберігаються в базі	ЕГ	76,67	9,17
		КГ	67,50	
6	Забезпечте ідентифікацію або авторизацію входу у вашу програму	ЕГ	78,33	10,83
		КГ	67,50	
7	Передбачте автоматичне відновлення даних у вашій програмі	ЕГ	75,83	11,67
		КГ	64,17	
8	Побудуйте інфологічну модель, визначте обмеження на атрибути та сутності, визначте зв`язки між сутностями. При виборі сутностей обґрунтуйте їхню необхідність. Сформулюйте всі обмеження природною мовою. При визначенні зв`язків необхідно обґрунтовувати вид зв`язку, його потужність, обов`язковість	ЕГ	78,33	7,50
		КГ	70,83	
9	Здійсніть декомпозицію загальної задачі та визначте групи користувачів у вашій системі та виконувані ними функції	ЕГ	78,33	10,83
		КГ	67,50	
10	Визначте кількість і види додатків, які потрібно розробити для реалізації загальної поставленої задачі у вашій системі. Задайте функціональне призначення кожного додатка	ЕГ	79,17	8,33
		КГ	70,83	
11	Виберіть найбільш характерні функції для більшості додатків і задайте їх як збережені процедури. Задайте словесний алгоритм виконання збережених процедур	ЕГ	80,83	11,67
		КГ	69,17	
12	Визначте необхідні тригери в системі. Задайте їхнє призначення та функціональні дії, що вимагаються	ЕГ	79,17	9,17
		КГ	70,00	
13	Для одного вибраного додатка здійсніть проектування, визначивши множину станів і їхнє відображення в термінах інтерфейсу користувача та множину переходів із зазначенням сигналів впливів, які викликають ці переходи. Задайте в умовних позначеннях зовнішній вигляд екранних форм для цього додатка	ЕГ	76,67	6,67
		КГ	70,00	
14	Проведіть критичне тестування вашої розробки програмної системи	ЕГ	79,17	7,50
		КГ	71,67	

Якщо погодитися з тим, що виконані та частково виконані завдання задовольняють замовника, то відсоток завдань, які можна вважати задовільними у командах, із врахуванням психотипів становитиме 11 завдань 83,29% (887 сумарних балів), а без врахування психотипу 8 завдань 66,02% (614 сумарних балів). Отже, можемо обчислити сумарний ефект від запропонованої гіпотези з формування команд розробників програмних Web-систем з використанням хмар-

ної технології із врахуванням психотипів, залучаючи того чи іншого члена відносно традиційних методів формування команд:

$$\Delta = \sum grup_u - \sum traditional_u \tag{6.4}$$

де Δ – приріст, який показує ефективність використання цієї методики формування груп розробників програмних Web-систем; $\sum grup_u$ – сума проєктів, які виконували команди та які були успішно або частково успішно виконані, формування яких відбувалося з огляду на психотипи її членів; $\sum traditional_u$ – сума проєктів, які виконували команди та які були успішно або частково успішно виконані, формування яких відбувалося за традиційними підходами.

Рисунок 6.15 – Аналіз виконання завдань з врахуванням та без врахування психотипу членів при формуванні команд

На основі емпірично визначених відсотків успішних та частково успішних розробок визначимо ефективність запропонованого методу, а саме обчисливши, згідно з формулою, різницю у відсотковому вимірі успішно виконані розробки програмних Web-систем з викорис-

танням хмарної технології командами, які формувалися із врахуванням психотипів кожного із членів команд розробників та без нього, відповідно до математичного виразу (6.4). Відображення отриманого результату наведено рис. 6.16.

Рисунок 6.16 – Оцінка ефективності виконання завдань з врахуванням та без врахування психотипу членів при формуванні команд

Отже, отримано загальний приріст ефективності успішно та частково розроблених програмних Web - систем з використанням хмарної технології, який становить 17,27 %. Він засвідчує зростання ефективно виконаних поставлених завдань.

Також ми проаналізували часткові випадки розроблення програмної системи з використанням Хмарної технології у гендерному розрізі, обчисливши згідно формули різницю (у відсотках) успішно та частково успішних робіт, які виконувались членами сформованих за різними підходами, застосовуючи, зокрема, психотипний підхід (рис. 6.17).

Рисунок 6.17 – Розподіл успішних розробок за способом формування команд (гендерний розподіл)

Отриманий результат показує, наскільки якість виконання проектних завдань із розроблення програмних систем використанням хмарної технології у моногамних групах, сформованих із врахуванням психотипів її членів вища, аніж у групах, створених на традиційних засадах. Різниця становить 11,77%.

Проводячи експеримент із застосування методів групової динаміки на ефективність розроблення програмних систем з використанням хмарної технології, ми розробили та запропонували методологію формування команд розробників із визначенням психотипу їхніх членів та з огляду на гендерний аспект. Її використовують для фірм, які спеціалізуються на розробці програмних систем, забезпечуючи збалансовану роботу команди та зменшення часового інтервалу на адаптацію кожного із членів у команді й швидкого пристосування до розподілу ролей при розробленні програмних систем, завдяки чому створюється позитивний мікроклімат та оптимізуються фінансові ресурси.

Висновки до шостого розділу

У дослідженні запропоновано модель визначення особистих рис (психотипів) кандидатів у члени команд розробників програмних систем та їхньої взаємодії у проєктній групі методами групової динаміки, з метою пом'якшення наслідків конфліктних ситуацій у команді, пришвидшення розроблення програмної системи з мінімальними фінансовими витратами.

Розроблено автоматизовану адаптивну систему тестування для визначення психотипу кожного із претендентів у члени команд розробників програмних систем та фахової компетентності для забезпечення якісного початкового формування команд з врахуванням визначених психотипів їх членів, завдяки чому унеможливлене потрапляння в команду двох членів з керівними психотипами і більше.

Проведено кластерний аналіз поставлених завдань командами розробників програмних систем у розрізі виконання підзадач як експериментальними групами, так і тими, які були сформовані за традиційним підходом (контрольні групи). Визначено ефективність запропонованої моделі формування команд розробників програмних систем з врахуванням психотипів їхніх членів згідно з експериментальними даними, що становить приріст на 10,21% при реалізації контрольних підзавдань.

Проведено системний аналіз та математичні розрахунки виконання поставлених завдань у процесі проведення експериментальних даних у розрізі двох закладів вищої освіти для підтвердження достовірності експерименту. Для цього розроблено та запропоновано критерії оцінки виконання поставлених завдань із розроблення програмних систем з використанням хмарної технології: більше 90% виконання – успішні, 60% – 90% – частково успішні та менше 60% – не успішні.

Проведено когнітивний аналіз виконання поставлених завдань для:

- контрольних груп: успішні – 18,34%, частково успішні – 47,68% та не успішні – 34,98%;
- експериментальних груп: успішні – 29,12%, частково успішні – 54,17% та не успішні – 18,74%.

Якщо врахувати успішні та частково успішні виконанні завдання за контрольними та експериментальними групами, які можна вважати

задовільними, то загальний показник експериментальних групах становить 83,29%, а в контрольних групах 66,02%.

Аналітичні дослідження та проведений експеримент підтвердили гіпотезу про доцільність визначення та врахування психотипу члена команди розробників програмних систем при формуванні команд, що забезпечило приріст ефективності виконання поставлених завдань на 17,27%. Отже, можна припустити, що приблизно 15% часу контрольні команди витратили на узгодження своєї позиції та місця у команді на початку виконання завдання з розроблення програмних систем.

Проведено аналітичні дослідження та порівняльний аналіз ефективності виконання поставлених завдань командами розробників за різних способів їхнього формування. Проаналізовано гендерний вплив на процес ефективної взаємодії команд розробників програмних систем з огляду на психотип членів команд, які були сформовані як змішані (71,52%) та моногамні (лише чоловіки або жінки) , що (59,75%). Дослідження підтвердило ефективність формування змішаних команд у гендерному розрізі із врахуванням психотипу їх членів розробників програмних систем з використанням хмарної технології, (на 11,77% вища, ніж у моногамних командах). Отже, зазначимо, що значно краще використовувати змішані групи розробників програмних систем та заздалегідь визначати і враховувати психотипи претендентів у команди розробників ще на початковому етапі водночас зменшуючи ймовірність виникнення декількох центрів впливу на прийняття рішень у команді розробників.

ВИСНОВКИ

У дисертації вирішено актуальну науково-прикладну проблему ефективного формування команд розробників програмних систем з огляду на їхні психотипи для скорочення адаптаційного часового інтервалу кожного із членів команди та швидкого знаходження своєї ролі підчас розроблення програмних проєктів із використанням хмарної технології, а також даний підхід забезпечує досягнення максимальної взаємодії у команді при виконання поставлених завдань та мінімальними фінансовими ресурсами.

Під час дослідження отримано такі наукові та практичні результати:

Проведено системний аналіз життєвих циклів інформаційних технологій (програмних технологій, баз та банків даних) у розрізі актуальності та перспективності використання під час розроблення програмних систем із використанням хмарної технології для мінімізації бюджетних витрат на розробку та функціонування інформаційної системи.

У дисертації вперше розроблено модель Енеограми визначення особистих рис (психотипів) кандидатів у члени команд розробників та їхній вплив на проблеми реалізації поставлених завдань, стратегії пом’якшення конфліктних наслідків у галузі інженерії програмних систем. У результаті емпіричних досліджень визначено найбільш важливі методи групової динаміки взаємодії команди розробників програмних систем та акцентовані наслідки пов’язані з використанням цих методів.

Вперше розроблено модель формування команд розробників програмних систем з огляду на психотипи членів команд у розрізі гендерного та національного різноманіття для забезпечення позитивного мікроклімату й ефективної роботи, а також зменшення ймовірності виникнення декількох центрів прийняття управлінських рішень.

Вперше розроблено структуру та реалізовано використання прикладного програмного забезпечення (проєктний менеджмент) як хмарного сервісу для управління розробкою програмної системи, що дало змогу отримати повноцінний доступ до розгорнутого сервісу на незалежній обчислювальній платформі із будь-якого місця та у будь-який час.

Вперше розроблено та запропоновано модель адаптивного тестування для визначення фахової компетенції претендентів у члени ко-

манд розробників програмних систем із використанням хмарної технології, що дало змогу адаптивно формувати послідовність подання тестового матеріалу відповідно до поточних інноваційних потреб сучасного ІТ-ринку.

Удосконалено та реалізовано математичні моделі групової динаміки і взаємодії членів команд під час формування та функціонування групи розробників програмних систем із використанням хмарної технології. Ці моделі змогу можливість пояснити спостережувані явища і зв'язки між ними та прогнозувати майбутній розвиток процесів стосунків між членами команд розробників програмних систем, що дає змогу вибрати оптимальні варіанти розвитку.

Удосконалено сіткові моделі та алгоритми архітектури високонавантажувальної розподіленої відмовостійкої Web-системи, основними відмінностями якої є можливість агрегування та спільного використання великих наборів гетерогенних обчислювальних ресурсів з опрацювання інформаційних потоків даних, розподілених між географічно розділеними територіями. Запропонована модель та алгоритми дають змогу використати додаткові сіткові ресурси, які підключені до функціональної сітки на відміну від традиційних підходів, там де ці ресурси не є доступні у межах одного обчислювального вузла на незалежній обчислювальній платформі.

Удосконалено модель структури групових ефектів та взаємодію між ними у командах, що забезпечує інтеграцію індивідуальних дій у спільній груповій діяльності під час розроблення програмних систем із використанням хмарної технології.

Отримали подальший розвиток інноваційні підходи до побудови високонавантажувальних відмовостійких розподілених кластерних програмних систем, що забезпечує суттєве збільшення сумарного обсягу інформаційних потоків даних вузлової та міжмодульної комунікації системи у загалом. Тому використання цього підходу є доцільним для розподілених відмовостійких програмних систем, де оперативна втрата інформаційних потоків даних є недопустима.

Отримали подальший розвиток алгоритми автоматизованого регулювання навантаженням на незалежних обчислювальних платформах інформаційних потоків даних для ефективного забезпечення масштабування (кластеризації) програмних систем

Практичне значення отриманих результатів полягає у створенні адаптивної автоматизованої системи тестування для визначення психотипу кожного із претендентів у члени команд розробників програ-

мних систем та фахової компетентності для забезпечення якісного початкового формування команд з огляду на визначені психотипи їхніх членів, завдяки якому до команди не потраплять двоє та більше учасників із керівними психотипами.

Дослідження та когнітивний аналіз виконання поставлених завдань підтвердили практичну цінність розроблення моделі формування команд розробників програмних систем з огляду на психотипи їх членів, що в відсотках становить:

- контрольних груп: успішні – 18,34 %, частково успішні – 47,68 % та не успішні – 34,98 %;
- експериментальних груп: успішні – 29,12 %, частково успішні – 54,17 % та не успішні – 18,74 %.

Проаналізувавши успішні та частково успішні виконані завдання за контрольними та експериментальними групами, які можна вважати задовільними, то у експериментальних групах цей показник становить 83,29 %, а у контрольних групах 66,02 %.

На основі розроблених моделей й методів підтверджено гіпотезу про доцільність визначення та врахування психотипу розробника програмних систем під час формування команд. Врахування психотипу членів команд цей же показник забезпечив приріст ефективності виконання поставлених завдань на 17,27 %. Отже, можна припустити, що приблизно 15 % часу контрольні команди витратили на узгодження своєї позиції та місця у на початку співпраці.

Проведено аналітичні дослідження та порівняльний аналіз ефективності виконання поставлених завдань командами розробників за різних способів їхнього формування. Проаналізовано гендерний вплив на процес ефективної взаємодії команд розробників програмних систем з огляду на психотип членів команд, які були сформовані, як з мішані (показник успішності – 71,52 %) та моногамні тобто лише чоловіки або жінки (показник успішності – 59,75 %). Отже, дослідження підтвердило ефективність формування змішаних команд у гендерному розрізі з огляду на психотип розробників програмних систем із використанням хмарної технології, вона на 11,77 % вища, ніж у моногамних командах. Отож, зазначимо, що значно краще використовувати змішані групи розробників програмних систем та заздалегідь визначати і враховувати психотипи претендентів у команди розробників ще на початковому етапі, під час формування команд, водночас зменшуючи ймовірність виникнення декількох центрів впливу на прийняття рішень у команді розробників.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. A. H. Eagly, and L. L. Carli, «The female leadership advantage: An evaluation of the evidence» Leadership Quarterly, 14, 807–834, 2003.

2. A. Nguyen-Duc, S. Khodambashi, J. A. Gulla, J. Krogstie, P. Abrahamsson , «Female Leadership in Software Projects—A Preliminary Result on Leadership Style and Project Context Factors» In: Kosiuczenko P., Madeyski L. (eds) Towards a Synergistic Combination of Research and Practice in Software Engineering. Studies in Computational Intelligence, vol 733, 2018.

3. A. Teh, E. Baniassad, D. V. Rooy, and C. Boughton, «Social Psychology and Software Teams: Establishing Task-Effective Group Norms,» IEEE Software, vol. 29, no.4, pp.53–58, Jul. 2012.

4. A. Trendowicz and J. Münch, «Factors Influencing Software Development Productivity—State-of-the-Art and Industrial Experiences,» Advances in computers, vol. 77, pp. 185-241, Dec. 2009.

5. Abomhara, M. (2015). Cyber security and the internet of things: Vulnerabilities, threats, intruders and attacks. Journal of Cyber Security and Mobility, 4(1), 65–88. doi:10.13052/jcsm2245-1439.414.

6. Ahn YY, Bagrow JP, Lehmann S. 2010 Link communities reveal multiscale complexity in networks. Nature 466, pp. 761–764. (doi:10.1038/nature09182).

7. Alali A, Kagdi H, Maletic J. 2008What's a typical commit? A characterization of open source software repositories. In Proc. of the 16th IEEE Int. Conf. On Program Comprehension, 2008, pp. 182–191. New York, NY: IEEE.

8. Al-Badarneh A, Al-Badarneh O. Challenges and interesting research directions in model driven architecture and data warehousing: a survey. Int J Comput Sci Inf Secur. 2016; 14(3): 364–98.

9. Albrecht AJ, Gaffney Jr JE. 1983 Software function, source lines of code, and development effort prediction: a software science validation. IEEE Trans. Softw. Eng. 9, pp. 639–648. (doi:10.1109/TSE.1983. 235271).

10. Alhazmi, O. H., Woo, S. W., & Malaiya, Y. K. (2006). Security vulnerability categories in major software systems. In Communication, Network, and Information Security (138–143).

11. Amro Al-Said Ahmad & Peter Andras (2019) Cloud-based software services delivery from the perspective of scalability, International

Journal of Parallel, Emergent and Distributed Systems, DOI: 10.1080/17445760.2019.1617864.

12. Andersson C, Read D. 2014 Group size and cultural complexity. Nature 511, E1. (doi:10.1038/nature13411).

13. Ariyachandra T, Watson H. Key organizational factors in data warehouse architecture selection. Decis Support Syst. 2010; 49(2): 200–12. doi:10.1016/j.dss.2010.02.006.

14. Armando, A., Merlo, A., Migliardi, M., & Verderame, L. (2012). Would you mind forking this process? A denial of service attack on Android (and some countermeasures). In IFIP International Information Security Conference pp. 13–24. Springer, Berlin, Heidelberg. doi: 10.1007/978-3-642-30436-1_2.

15. Atashzar, H., Torkaman, A., Bahrololum, M., & Tadayon, M. H. (2011). A survey on web application vulnerabilities and countermeasures. In Computer Sciences and Convergence Information Technology (ICCIT), 2011 6th International Conference on pp. 647–652.

16. Atzeni P, Cheung D, Ram S. Conceptual modeling: 31st International conference, ER 2012, Florence, Italy, October 15-18, 2012: Proceedings. Heidelberg: Springer; 2012. pp. 64–77.

17. Ayman, R. (2000). Impact of team diversity on collaboration dynamics. Paper presented at the Collaborating across Professional Boundaries Conference. Retrieved April 25, 2011, from http://www.stuart.iit.edu/ipro/papers/pdf/
ayman.pdf.

18. B. Barry, «Centre for Systems and Software Engineering,» 2012. from http://sunset.usc.edu/csse/research/COCOMOII/cocomo_main.html. [retrieved: August, 2017].

19. B. Tessem and F. Maurer, «Job Satisfaction and Motivation in a Large Agile Team», In International Conference on Extreme Programming and Agile Processes in Software Engineering, Springer Heidelberg, vol. 4536, pp. 54–61., 2007.

20. Baboo SS, Kumar PR. Next generation data warehouse design with OLTP and OLAP systems sharing the same database. Int J Comput Appl. 2013; 72(13): doi:10.5120/12557-9282.

21. Bai, X., Xing, L., Zhang, N., Wang, X., Liao, X., Li, T., & Hu, S. M. (2016). Staying secure and unprepared: Understanding and mitigating the security risks of Apple ZeroConf. In Security and Privacy (SP), 2016 IEEE Symposium on pp. 655–674. doi: 10.1109/SP.2016.45.

22. Barnum, C. M. (2000). Building a team for usercentred design. In Proceedings of IEEE Professional Communication Society International Professional Communication Conference and Proceedings of the 18th Annual ACM International Conference on Computer Documentation: Technology & Teamwork pp. 325-332. Piscataway, NJ: IEEE Press.

23. Barsade, S. G. 2002. "The Ripple Effect: Emotional Contagion and Its Influence on Group Behavior." Administrative Science Quarterly 47 (4): pp. 644–675. doi:10.2307/3094912.

24. Barsade, S. G., and A. P. Knight . 2015. "Group Affect." Annual Review of Organizational Psychology and Organizational Behavior 2 (1):pp. 21–46. doi:10.1146/annurev-orgpsych-032414-111316.

25. Barsade, S. G., and D. E. Gibson . 2012. "Group Affect Its Influence on Individual and Group Outcomes." Current Directions in Psychological Science 21 (2): pp. 119–123. doi:10.1177/0963721412438352.

26. Barsade, S. G., and D. E. Gibson. 1998. "Group Emotion: A View from Top and Bottom" In Research on Managing Groups and Teams, edited by D. Gruenfeld, E. Mannix, and M. Neale, pp. 81–102. Stamford, CT: JAI Press.

27. Bartel, C. A., and R. Saavedra . 2000. "The Collective Construction of Work Group Moods." Administrative Science Quarterly 45 (2): pp. 197–231. doi:10.2307/2667070.

28. Bartlett, M. S., G. C. Littlewort , M. G. Frank , C. Lainscsek , I. R. Fasel , and J. R. Movellan . 2006. "Automatic Recognition of Facial Actions in Spontaneous Expressions." Journal of Multimedia 1 (6): pp. 22–35. doi:10.4304/jmm.1.6.22-35.

29. Basri, S. «Software Process Improvement in Very Small Entities», PhD Thesis, Dublin City University, Ireland. 2010.

30. Basri, S., & O' Connor, R. V. (2010, May 25-27). Evaluation of knowledge management process in very small software companies: A survey. In Proceedings of the 5th International Conference on Knowledge Management, Kuala Terengganu, Terengganu, Malaysia pp. 1-6.

31. Bass, B.M. and Dunteman, G. «Behaviour in Groups as a Function of Self, Interaction and Task Orientation.» Journal of Abnormal Social Psychology. Vol. 66, Num. 4, 1963, pp 19 – 28.

32. Bates, D. M., M. Mächler, B. Bolker, and S. Walker. 2014. "Fitting Linear Mixed-Effects Models Using lme4." Journal of Statistical Software 67 (1): pp. 1–48. doi:10.18637/jss.v067.i01.

33. Batini C, Cappiello C, Francalanci C, Maurino A. Methodologies for data quality assessment and improvement. ACM Comput Surv (CSUR). 2009; 41(3): pp. 1–52. doi:10.1145/1541880.1541883.

34. Batisha, J. and de Figueiredo, A.D., 2000,' SPI in a Very Small Team: A Case with CMM', Software Process Improvement and Practice, Vol. 5, No. 4, pp. 243-255. doi:10.1002/1099-1670(200012)5:4<243::AIDSPIP126>3.0.CO.

35. Beaver, J. M. and Schiavone G. A., 2006. «The effects of development team skill on software product quality». ACM SIGSOFT Software Engineering Notes, Vol. 31, No. 3. pp. 1–5. doi:10.1145/1127878.1127882.

36. Berndt, D. J., and J. Clifford. 1994. "Using Dynamic Time Warping to Find Patterns in Time Series." Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining (AAAIWS'94) pp. 359–370.

37. Bettany, A., & Halsey, M. (2017). Windows virus and malware troubleshooting, edited by Andrew Bettany, Mike Halsey,North Yorkshire, UK: Apress, pp 21–39.

38. Bhavani, A. B. (2013). Cross-site scripting attacks on android webview. arXiv preprint arXiv:1304.7451.

39. Bin Basri, S and O' Connor, RV., 2010 «Organizational commitment towards software process improvement an irish software VSEs case study», Information Technology (ITSim), 2010 International Symposium, 15-17 June 2010, Kuala Lumpur.

40. Bird C, Pattison D, D'Souza R, Filkov V, Devanbu P. 2008 Latent social structure in open source projects. In Proc. of the 16th ACM SIGSOFT Int. Symp. On Foundations of Software Engineering. SIGSOFT '08/FSE-16, pp. 24–35. New York, NY: ACM.

41. Bleeping Computer, 2017 (January 15). Android Was 2016's most vulnerable product. Retrieved from https://www.bleepingcomputer.com/ news/security/android-was-2016s-most-vulnerable-product.

42. Blickensderfer, E., J. A. Cannon-Bowers, and E. Salas. 1998. "Cross-Training and Team Performance." In Making Decisions Under Stress: Implications for Individual and Team Training, edited by J. A. Cannon-Bowers and E. Salas, pp. 299–311. Washington, DC: American Psychological Association.

43. Bliese, P. D. 2000. "Within-Group Agreement, Non-Independence, and Reliability: Implications for Data Aggregation and

Analysis.” In Multilevel Theory, Research, and Methods in Organizations, edited by K. J. Klein and S. W. J. Kozlowski, pp. 349–381. San Francisco, CA: Jossey-Bass.

44. Boehner, K., R. DePaula, P. Dourish, and P. Sengers. 2007. “How Emotion Is Made and Measured.” International Journal of Human-Computer Studies 65 (4): pp. 275–291. doi:10.1016/j.ijhcs.2006.11.016.

45. Bohorquez JC, Gourley S, Dixon AR, Spagat M, Johnson NF. 2009 Common ecology quantifies human insurgency. Nature 462, pp. 911–914. (doi:10.1038/nature08631).

46. Brave, S., and C. Nass. 2003. “Emotion In Human-computer Interaction.” In The Human-Computer Interaction Handbook: Fundamentals, Evolving Technologies, and Emerging Applicationsis, edited by J. A. Jacko and A. Sears, pp. 81–96. Mahwah, NJ: Lawrence Erlbaum Associates, Inc.

47. Breur T. Data quality is everyone’s business – designing quality into your data warehouse – part 1. J Dir Data Dig Mark Pract. 2009; 11(1): pp. 20–29. doi:10.1057/dddmp.2009.14.

48. Brown, N. M., and C. R. Moren. 2003. “Background Emotional Dynamics of Crew Resource Management: Shame Emotions and Coping Responses.” International Journal of Aviation Psychology 13 (3): pp. 269–286. doi:10.1207/S15327108IJAP1303_05.

49. Butler, E. A. 2011. “Temporal Interpersonal Emotion Systems the “TIES” That Form Relationships.” Personality and Social Psychology Review 15 (4): pp. 367–393. doi:10.1177/1088868311411164.

50. C. D. O. Melo, D. S. Cruzes, F. Kon, and R. Conradi, «Interpretative case studies on agile team productivity and management,» Information and Software Technology, vol. 55, pp. 412-427, Feb.2013.

51. C.O. Melo, «Productivity of agile teams: an empirical evaluation of factors and monitoring processes,» Ph.D. dissertation, Universidade de São Paulo, 2015.

52. Calero C, Piattini M, Genero M. Metrics for controlling database complexity. Dev Qual Complex Data. 2000: pp. 48–68.

53. Cebula, J. L., & Young, L. R. (2010). A taxonomy of operational cyber security risks (Research report No. CMU/SEI-2010-TN-028). Carnegie-Mellon Univ Pittsburgh Pa Software Engineering Inst.

54. Chan, D. 1998. “Functional Relations among Constructs in the Same Content Domain at Different Levels of Analysis: A Typology of Composition Models.” Journal of Applied Psychology 83 (2): pp. 234–246. doi:10.1037/0021-9010.83.2.234.

55. Chen, D. D., Woo, M., Brumley, D., & Egele, M. (2016). Towards automated dynamic analysis for linux-based embedded firmware. In Network and Distributed System Security Symposium (NDSS). DOI: 10.14722/ndss.2016.23415.

56. Chen, Y., Khandaker, M., & Wang, Z. (2017). Pinpointing vulnerabilities. In Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security pp. 334–345. ACM. doi: 10.1145/3052973.3053033.

57. Cohen, S. G. and Bailey, D. E., , «What Makes Teams Work: Group effective Research from The Shop Floor to the Executive Suite», Journal of Management, Vol. 23 No. 3, pp 234-256. 1997.

58. Cohen, S., & Bailey, D. (1997). What makes teams work: Group effectiveness research from the shop floor to the executive suite. Journal of Management, 23(3), pp. 239–290. doi:10.1177/014920639702300303.

59. Coleman, G. and O'Connor, R. V., «The influence of managerial experience and style on software development process». International Journal of Technology, Policy and Management. Vol. 8, No. 1, 2008.

60. Collins, A. L., S. A. Lawrence, A. C. Troth, and P. J. Jordan. 2013. "Group Affective Tone: A Review and Future Research Directions." Journal of Organizational Behavior 34 (S1): pp.43–62. doi:10.1002/job.1887.

61. Contractor N. S., DeChurch L. A., Carson J., Carter D. R., Keegan B. 2012 The topology of collective leadership. Lead. Q. 23, pp. 994–1011. (doi:10.1016/j.leaqua.2012.10.010).

62. Courtemanche C, Carden A. Competing with costco and sam's club: warehouse Club entry and grocery prices. Southern Economic Journal. 2014;80(3): pp. 565–85. doi:10.4284/0038-4038-2012.135.

63. Cui, B., Liang, X., & Wang, J. (2011). The Study on Integer Overflow Vulnerability Detection in Binary Executables Based Upon Genetic Algorithm. In Foundations of Intelligent Systems edited by Yinglin Wang, Tianrui Li, pp. 259–266. Berlin: Springer. doi:10.1007/978-3-642-25664-6_30.

64. CVE Details, 2017 (January 19). Top 50 Products By Total Number Of "Distinct" Vulnerabilities in 2016. Retrieved from https://www.cvedetails.
com/top-50-products.php?year=2016.

65. D. S. Cruzes and T. Dyba, «Recommended Steps for Thematic Synthesis in Software Engineering», 2011 International Symposium on Empirical Software Engineering and Measurement (ESEM '11), 2011.

66. D'orazio, C. J., Lu, R., Choo, K. K. R., & Vasilakos, A. V. (2017). A Markov adversary model to detect vulnerable iOS devices and vulnerabilities in iOS apps. Applied Mathematics and Computation, 293, pp. 523–544. doi:10.1016/j.amc.2016.08.05.

67. Dedić N, Stanier C. An Evaluation of the Challenges of Multilingualism in Data Warehouse Development. Conference: 18th International Conference on Enterprise Information Systems - ICEIS 2016; 2016; At Rome, Italy, vol. 1. doi: 10.5220/0005858401960206.

68. Delmar F, Shane S. 2006 Does experience matter? The effect of founding team experience on the survival and sales of newly founded ventures. Strateg. Organ. 4, pp. 215–247. (doi:10.1177/1476127006066596).

69. Denis JL, Lamothe L, Langley A. 2001 The dynamics of collective leadership and strategic change in pluralistic organizations. Acad. Manage. J. 44, pp. 809–837. (doi:10.2307/3069417).

70. Derex M, Beugin MP, Godelle B, Raymond M. 2013 Experimental evidence for the influence of group size on cultural complexity. Nature 503, pp. 389–391. (doi:10.1038/nature12774).

71. Details, C. V. E., (2016b, December 21). Vulnerability Details: CVE-2016-3245. Retrieved from https://www.cvedetails.com/cve/CVE-2016-3245.

72. Dimberg, U., M. Thunberg, and K. Elmehed. 2000. "Unconscious Facial Reactions to Emotional Facial Expressions." Psychological Science 11 (1): pp. 86–89. doi:10.1111/1467-9280.00221.

73. Dinh, H. T., Lee, C., Niyato, D., & Wang, P. (2013). A survey of mobile cloud computing: Architecture, applications, and approaches. Wireless Communications and Mobile Computing, 13(18), pp. 1587–1611. doi:10.1002/wcm.1203.

74. D'Mello, S., and R. A. Calvo. 2013. "Beyond the Basic Emotions: what Should Affective Computing Compute?" Proceedings of CHI'13 Extended Abstracts on Human Factors in Computing Systems pp. 2287–2294.

75. Elkins, A. N., E. R. Muth, A. W. Hoover, A. D. Walker, T. L. Carpenter, and F. S. Switzer. 2009. "Physiological Compliance and Team Performance." Applied Ergonomics 40 (6): pp. 997–1003. doi:10.1016/j.apergo.2009.02.002.

76. Ellis, A. P. J., and M. J. Pearsall. 2011. "Reducing the Negative Effects of Stress in Teams through Cross-Training: A Job Demands-

Resources Model.” Group Dynamics: Theory, Research, and Practice 15 (1): pp. 16–31. doi:10.1037/a0021070.

77. F. Nasirzadeh and P. Nojedehi, «Dynamic modelling of labour productivity in construction projects,» International Journal of Project Management, vol. 31, no. 6, Aug. 2013, pp. 903-911.

78. Farahmand, F., Navathe, S. B., Enslow, P. H., & Sharp, G. P. (2003). Managing vulnerabilities of information systems to security incidents. In Proceedings of the 5th international conference on Electronic commerce pp. 348–354. ACM. doi: 10.1145/948005.948050.

79. Faraj S, Sproull L. 2000 Coordinating expertise in software development teams. Manage. Sci. 46, pp. 1554–1568. (doi:10.1287/mnsc.46.12.1554.12072).

80. Fazzinga B, Flesca S, Masciari E, Furfaro F. Efficient and effective RFID data warehousing. Proceedings of the 2009 International Database Engineering & Applications Symposium on - IDEAS ’09; 2009. doi:10.1145/1620432.1620459.

81. Fendt J, Sachs W. Grounded theory method in management research. Organ Res Method. 2008;11(3): pp. 430–55. doi:10.1177/1094428106297812.

82. Feng, H., & Shin, K. G. (2016). Understanding and defending the Binder attack surface in Android. In Proceedings of the 32nd Annual Conference on Computer Security Applications pp. 398–409. ACM. doi: 10.1145/2991079.2991120.

83. Fournaris, A. P., PoceroFraile, L., & Koufopavlou, O. (2017). Exploiting hardware vulnerabilities to attack embedded system devices: A survey of potent microarchitectural attacks. Electronics, 6(3), 52. doi:10.3390/electronics6030052.

84. Fredrickson, B. L., and C. Branigan. 2005. “Positive Emotions Broaden the Scope of Attention and Thought-Action Repertoires.” Cognition and Emotion 19 (3): pp. 313–332. doi:10.1080/02699930441000238.

85. Furumo, K. and Pearson, J.M., «An Empirical Investigation of how Trust, Cohesion and Performance Vary in Virtual and Face to Face Teams». System Sciences, Proceedings of the 39th Annual Hawaii International Conference, Vol. 1, pp. 26., 2006.

86. G. Wikstrand and J. Borstler. « Success factors for team project courses» In Proceedings of the 19th Conference on Software Engineering Education and Training. pp. 95–102, 2006.

87. Garg, S., & Baliyan, N. (2019). Data on vulnerability detection in android. Data in Brief, 22, pp. 1081–1087. doi:10.1016/j.dib.2018.12.038.

88. Gaur H, Sharma R. Assessment of data warehouse model quality. Int J Adv Res Comput Sci. 2013; 4(9). http://ezaccess.libraries.psu.edu/login?url=http://search.proquest.com.ezaccess.libraries.psu.edu/docview/1444033602?accountid=13158.

89. Ghapanchi AH, Aurum A, Low G. 2011 A taxonomy for measuring the success of open source software projects. First Monday 16. (doi:10.5210/fm. v16i8.3558).

90. Giorgino, T. 2009. "Computing and Visualizing Dynamic Time Warping Alignments in R: The Dtw Package." Journal of Statistical Software 31 (7): pp. 1–24. doi:10.18637/jss.v031.i07.

91. Girvan M, Newman MEJ. 2002 Community structure in social and biological networks. Proc. Natl Acad. Sci. USA 99, pp. 7821–7826. (doi:10.1073/pnas. 122653799).

92. Glick, W. H. 1985. "Conceptualizing and Measuring Organizational and Psychological Climate: Pitfalls in Multilevel Research." Academy of Management Review 10 (3): pp. 601–616. doi:10.5465/amr.1985.4279045.

93. Golfarelli M, Rizzi S. A comprehensive approach to data warehouse testing. Proceedings of the ACM twelfth international workshop on Data warehousing and OLAP pp. 17–24. Chicago, IL, USA, ACM; 2009 Nov.

94. Gorbachuk M., Lazor A., Bandyra V., Yurchak I. end Pasyeka M. Method and Parallelization Algorithms of Synthesis of Empirical Models Taking Into Account the Measurement Errors. CADSM 2015 Матеріали XIII Міжнародної науково-технічної конференції «Досвід розробки та застосування приладо-технологічних САПР в мікроелектроніці», м. Львів, 2015. С. 319–327.

95. Gorla, N and Lam, Y.W., «Who Should Work With Whom? Building Effective Software Project Teams», Communications of the ACM ,Vol. 47, Issue. 6, pp. 123. 2004.

96. Gosain A, Nagpal S, Sabharwal S. Quality metrics for conceptual models for data warehouse focusing on dimension hierarchies. ACM SIGSOFT Software Eng Note. 2011; 36(4): pp. 1–5. doi:10.1145/1988997.1989015.

97. Gosain A. Literature review of data model quality metrics of data warehouse. Procedia Comput Sci. 2015; 48: pp. 236–43. doi:10.1016/j.procs.2015.04.176.

98. Granebring A, Révay P. Service-oriented architecture is a driver for daily decision support. Kybernetes. 2007; 36(5): pp. 622–35. doi:10.1108/03684920710749712.

99. Gunes, H., and B. Schuller. 2012. “Categorical and Dimensional Affect Analysis in Continuous Input: Current Trends and Future Directions.” Image and Vision Computing 32 (2): pp. 120–136. doi:10.1016/j.imavis.2012.06.016.

100.H. Igaki, N. Fukuyasu, S. Saiki, S. Matsumoto, and S. Kusumoto, «Quantitative assessment with using ticket driven development for teaching Scrum framework» 36th Inter. Conf. on Software Engineering,, Hyderabad, India, pp. 372-381, 2014.

101.Hall, T., Beecham, S., Verner, J. and Wilson, D., 2008. «The Impact of Staff turnover on Software Project: The Importance of Understanding What makes Software Practitioners Tick», Proceedings of ACM SIGMIS CPR, ACM New York, pp. 30-39.

102.Hall, T., Rainer, A. and Baddoo, N. 2002, «Implementing Software Process Improvement: An empirical Study», Software Process, Improvement and Practice, Vol. 7, No 1, pp. 3-15.

103.Han J, Pei J, and Kamber M. Data mining: Concepts and techniques. Waltham, MA: Elsevier; 2012.

104.Han SC et al. Using MCRDR based Agile approach for expert system development. Computing. 2014: 96(9) pp. 897–908.

105.Haq QM. Data mapping for data warehouse design. Waltham, MA: Morgan Kaufmann; 2016. pp. 67–165.

106.Hart, S. G., and L. E. Staveland. 1988. “Development of NASA-TLX (Task Load Index): Results of Empirical and Theoretical Research.” Advances in Psychology 52: pp. 139–183. doi:10.1016/S0166-4115(08)62386-9.

107.Heiderich, M., 2012. Towards elimination of xss attacks with a trusted and capability controlled dom (Doctoral dissertation). University in Bochum, Germany 10.1094/PDIS-11-11-0999-PDN.

108.Henning, R. A., A. G. Armstead, and J. K. Ferris. 2009. “Social Psychophysiological Compliance in a Four-Person Research Team.” Applied Ergonomics 40 (6): pp. 1004–1010. doi:10.1016/j.apergo.2009.04.009.

109.Henrich J. 2004 Demography and cultural evolution: how adaptive cultural processes can produce maladaptive losses: the Tasmanian case. Am. Antiquity 69, pp. 197–214. (doi:10.2307/4128416).

110.Herndon AA, Cramer M, Nicholson T, Miller S. Analysis of Advanced Flight Management Systems (FMS), Flight Management Computer (FMC) field observations trials: area Navigation (RNAV) holding patterns. Digital Avionics Systems Conference (DASC), 2011 IEEE/AIAA 30th pp. 4A1–1. Seattle, WA, USA, IEEE; 2011, Oct.

111.Hoegl, M., & Proserpio, L. (2004). Team member proximity and teamwork in innovative projects. Research Policy, 3(8), pp. 1153–1165. doi:10.1016/j.respol.

112.Homaei, H., & Shahriari, H. R. (2017). Seven years of software vulnerabilities: The ebb and flow. IEEE Security & Privacy, (1), pp. 58–65. doi:10.1109/MSP.2017.15.

113.Honaker, J., G. King, and M. Blackwell. 2011. "Amelia II: A Program for Missing Data." Journal of Statistical Software 45 (7): pp. 1–47. doi:10.18637/jss.v045.i07.

114.Horbiychk M., Khrabatyn R., Bandura V., Khrabatyn O., Samaniv L. end Pasyeka M. Computer Modeling of Soil Contamination in Halych District Based on the Theory of Neural Networks. Международный научный журнал «Управляющие системы и машины», м.Київ: – №4(258), 2015, Київ, С. 83–85.

115.Horwitz SK, Horwitz IB. 2007 The effects of team diversity on team outcomes: a meta-analytic review of team demography. J. Manag. 33, pp. 987–1015. (doi:10.1177/0149206307308587).

116.Howard, L. W., & Foster, T. S. (1999). The influence of human resource practices on empowerment and employee perceptions of management commitment to quality. Journal of Quality Management, 4(1), pp. 5–22. doi:10.1016/S1084-8568(99)80093-5.

117.Hudson, T., & Rudolph, L. (2015). Thunderstrike: EFI firmware bootkits for apple macbooks. In Proceedings of the 8th ACM International Systems and Storage Conference (p. 15). ACM. doi: 10.1145/2757667.2757673.

118.Hwang MI, Xu H. A structural model of data warehousing success. J Comput Inf Syst. 2008;49(1): pp. 48–56. doi:10.1080/08874417.2008.11645305.

119.I. Fatema, «Agile teamwork productivity influence factors,» Jan. 2017. [Online] Available https://goo.gl/forms/I5xGdQGqFMk9he5f2. [retrieved: August, 2017].

120.Inbar, O., and N. Tractinsky. 2009. "The Incidental User." interactions 16 (4): pp. 56–59. doi:10.1145/1551986.1551998.

121.Inbar, O., and N. Tractinsky. 2012. “Lowering the Line of Visibility: Incidental Users in Service Encounters.” Behaviour and Information Technology 31 (3): pp. 245–260. doi:10.1080/0144929X.2011.563796.

122.Inmon W.H. Building the Data Warehouse Third Edition 2008 432c.

123.Insights Association, 2017 (July 23). Compound Annual Growth Rate (CAGR). Retrieved from https://www.insightsassociation.org/issues-policies/glossary/compound-annual-growth-rate-cagr.

124.Iqbal, M., & Rizwan, M. (2009). Application of 80/20 rule in software engineering Waterfall Model. In Information and Communication Technologies, 2009. ICICT’09. International Conference on pp. 223–228. IEEE. doi: 10.1109/ICICT.2009.5267186.

125.ISO/IEC DTR 29110-1, «Software Engineering - Lifecycle Profiles for Very Small Entities (VSE) -- Part 1: VSE profiles Overview». Geneva: International Organization for Standardization (ISO), 2011.

126.J. Borstler. «Experience with work-product oriented software development projects». Computer Science Education, vol 11(2), pp. 111–133, 2001.

127.J. C. H. Ellis, S. A. Demurjian, and J. F. Naveda. «Software Engineering: Effective Teaching and Learning Approaches and Practices» IGI Global, Hershey, NY, 2009.

128. J. M. Lyneis and D. N. Ford, «System dynamics applied to project management: a survey, assessment, and directions for future research,» System Dynamics Review, vol. 23, no. 2-3, pp. 157-189, Jun. 2007.

129.J. M. Verner, M. A. Babar, N. Cerpa, T. Hall, and S. Beecham, «Factors that motivate software engineering teams: A four country empirical study,». Journal of Systems and Software, vol. 92, June. 2014, pp. 115-127.

130.J. Vanhanen, T. O. A. Lehtinen, and C. Lassenius, «Software engineering problems and their relationship to perceived learning and customer satisfaction on a software capstone project», Journal of Systems and Software, vol. 137, pp. 50-66, 2018/03/01/, 2018.

131.Jaeger, B. C., L. J. Edwards, K. Das, and P. K. Sen. 2017. “An R2 Statistic for Fixed Effects in the Generalized Linear Mixed Model.” Journal of Applied Statistics 44 (6): pp. 1086–1105. doi:10.1002/sim.3429.

132.Jarvenpaa, S. L., T. R. Shaw, and D. S. Staples. 2004. “Toward Contextualized Theories of Trust: The Role of Trust in Global Virtual

Teams.” Information Systems Research 15 (3): 250–267. doi:10.1287/isre.1040.0028.

133.Javed, T., Maqsood, M. and Durrani Q., ‘A Survey to Examine the effect of Team Communication on Job satisfaction in Software Industry’, ACM SIGSOFT Software Engineering Notes, Vol. 29, No. 2, pp. 6.

134.Jeffrey W. Top 5 data warehouses on the market today; 2015 [accessed 2016 Dec 25]. Monitis. http://www.monitis.com/blog/top-5-data-warehouses-on-the-market-today.

135.Jian, J. Y., A. M. Bisantz, and C. G. Drury. 2000. “Foundations for an Empirically Determined Scale of Trust in Automated Systems.” International Journal of Cognitive Ergonomics 4 (1): pp. 53–71. doi: 10.1207/S15327566IJCE0401_04.

136.Joshi, J., & Parekh, C. (2016). Android smartphone vulnerabilities: A survey. In Advances in Computing, Communication, & Automation (ICACCA)(Spring), International Conference on pp. 1–5. IEEE. doi: 10.1109/ICACCA.2016.7578857.

137.K. Lewin, A dynamic theory of personality. New York: McGraw-Hill, 1935.

138.K. Petersen, «Measuring and predicting software productivity: A systematic map and review,» Information and Software Technology, vol. 53, pp. 317-343, Apr.2011.

139.Kalliamvakou E, Gousios G, Blincoe K, Singer L, German DM, Damian D. 2014 The promises and perils of mining GitHub. In Proc. of the 11thWorking Conf. on Mining Software Repositories, pp. 92–101. New York, NY: ACM.

140.Kaplan, S., K. LaPort, and M. J. Waller. 2013. “The Role of Positive Affectivity in Team Effectiveness during Crises.” Journal of Organizational Behavior 34 (4): pp. 473–491. doi:10.1002/job.1817.

141.Kara, M. (2012). Review on common criteria as a secure software development model. International Journal of Computer Science & Information Technology, 4(2), pp. 83. doi:10.5121/ijcsit.2012.4207.

142.Kelly, J. R., and S. G. Barsade. 2001. “Mood and Emotions in Small Groups and Work Teams.” Organizational Behavior and Human Decision Processes 86 (1): pp. 99–130. doi:10.1006/obhd.2001.2974.

143.Kenny, D. A., D. Kashy, and W. L. Cook. 2006. Dyadic Data Analysis. New York: Guilford Press.

144.Khajaria K, Kumar M. Modeling of security requirements for decision information systems. ACM SIGSOFT Software Eng Note. 2011;36(5):1. doi:10.1145/2020976.2020989.

145.Khrabatyn R., Bandura V., Khrabatyn O. end Pasyeka M. Decision-Making Support Systems in Agriculture. Algorithm, Principles of Construction and Functionality. Международный научный журнал «Управляющие системы и машины», м. Київ, №1(255), 2015, С. 80–86.

146.Kimball R, Ross M. The data warehouse. Indianapolis, IN: John Wiley & Sons; 2013.

147.Kimball R. The data warehouse lifecycle toolkit. 2nd ed. Indianapolis, IN: Wiley Pub; 2008. Chapter 1 & 2.

148.Kirkman, B. L., Rosen, B., Tesluk, P. E., & Gibson, C. B. (2004). The impact of team empowerment on virtual team performance: The moderating role of face-to-face interaction. Academy of Management Journal, 47(2), pp. 175–192. doi:10.2307/20159571.

149.Knight, A. P., and N. Eisenkraft. 2015. “Positive Is Usually Good, Negative Is Not Always Bad: The Effects of Group Affect on Social Integration and Task Performance.” The Journal of Applied Psychology 100 (4): pp. 1214–1227. doi:10.1037/apl0000006.

150.Koch S, Schneider G. 2002 Effort, co-operation and co-ordination in an open source software project: GNOME. Inf. Syst. J. 12, pp. 27–42 (doi:10.1046/j.1365-2575.2002.00110.x).

151.Kohavi R, Mason L, Parekh R, Zheng Z. Lessons and challenges from mining retail E-commerce data. Mach Learn. 2004; 57(1–2): pp. 83–113. doi:10.1023/B:MACH.0000035473.111.

152.Komiyama, T., Sunazuka, T., & Koyama, S. (2000). Software process assessment and improvement in NEC - current status and future direction. Software Process Improvement and Practice, 5(1), pp. 31–43. doi:10.1002/(SICI)1099-1670(200003)5:1<31::AID-SPIP109>3.0.CO;2-K.

153.Kugler, T., T. Connolly, and L. D. Ordóñez. 2012. “Emotion, Decision, and Risk: Betting on Gambles versus Betting on People.” Journal of Behavioral Decision Making 25 (2): pp. 123–134. doi:10.1002/bdm.724.

154.Kuznetsova, A., P. B. Brockhoff, and R. Christensen. 2012. “lmerTest: Tests for Random and Fixed Effects for Linear Mixed Effect Models (lmer Objects of lme4 Package)”.

155.L. L. R. Rodrigues, N. Dharmaraj, and B. R. Shrinivasa Rao, «System dynamics approach for change management in new product

development,» Management Research News, vol. 29, no. 6, Aug. 2006, pp. 512-523.

156.Laporte, C.Y., Alexandre, S., and O'Connor, R. 2008. «A Software Engineering Lifecycle Standard for Very Small Enterprises», R.V.O'Connor et al (Eds) Proceedings of EuroSPI Springer-Verlag, CCIS Vol. 16, pp. 129-141.

157.Larson, J. R. 2010. In Search of Synergy in Small Group Performance. New York: Psychology Press.

158.Lazer D et al. 2009 Life in the network: the coming age of computational social science. Science 323, pp. 721–723. (doi:10.1126/science.1167742).

159.LeBreton, J. M., and J. L. Senter. 2008. "Answers to 20 Questions about Interrater Reliability and Interrater Agreement." Organizational Research Methods 11 (4): pp. 815–852. doi:10.1177/1094428106296642.

160.Lemme S. Database management best practices: expert tips for improving DBMS infrastructures; 2006, Jul 16. http://searchdatamanagement.techtarget.com/news/1199349/Database-management-best-practices-Expert-tips-for-improving-DBMS-infrastructures.

161.Li H. Applications of data warehousing and data mining in the retail industry. Proceedings of ICSSSM '05. 2005 International Conference on Services Systems and Services Management; 2005. doi:10.1109/icsssm.2005.1500153.

162.Li, X., Chang, X., Board, J. A., & Trivedi, K. S. (2017). A novel approach for software vulnerability classification. In Reliability and Maintainability Symposium (RAMS), 2017 Annual pp. 1–7. doi: 10.1109/RAM.2017.7889792.

163.Littlepage, G.E., Cowart, L. and Kerr, B., 1989. «Relationships between Group Environment Scales and Group Performance and Cohesion», Small Group Research, Vol. 20, No. 1, pp. 50-61.

164.Littlewort, G. C., J. Whitehill, T. L. Wu, I. R. Fasel, M. G. Frank, J. R. Movellan, and M. S. Bartlett. 2011. "The Computer Expression Recognition Toolbox (CERT)." Paper presented at the 2011 IEEE International Conference on Automatic Face and Gesture Recognition and Workshops (FG 2011), Santa Barbara, CA, USA.

165.Littlewort, G. C., M. S. Bartlett, I. R. Fasel, J. Chenu, T. Kanda, H.1 Ishiguro, and J. R. Movellan. 2003. "Towards Social Robots: Automatic Evaluation of Human-Robot Interaction by Face Detection and Expression Classification." Proceedings of the 16th International

Conference on Neural Information Processing Systems (NIPS'03) pp. 1563–1570.

166.Lottridge, D., M. Chignell, and A. Jovicic. 2011. “Affective Interaction: Understanding, Evaluating. And Designing for Human Emotion.” Reviews of Human Factors and Ergonomics 7 (1): pp. 197–217. doi:10.1177/1557234X11410385.

167.Lu X, Huang L, Heng MS. Critical success factors of inter-organizational information systems: a case study of Cisco and Xiao Tong in China. Inf Manag. 2006; 43(3): pp. 395–408. doi:10.1016/j.

168.Luna-Reyes LF, Andersen DL. Collecting and analyzing qualitative data for system dynamics: methods and models. Syst Dyn Rev. 2003;19(4):
pp. 271–96. doi:10.1002/(ISSN)1099-1727.

169.M. A. West, «Reflexivity and work group effectiveness: A conceptual integration» In M. A. West (Ed.), Handbook of Work Group Psychology pp. 555-579. Chichester: John Wiley & Sons Ltd, 1996.

170.M. J. Mawdesley & S. Al-Jibouri, «Modelling construction project productivity using systems dynamics approach,» International Journal of Productivity and Performance Management, vol. 59, no.1, Dec. 2009, pp. 18-36.

171.Malabocchia F, Buriano L, Mollo M, Richeldi M, Rossotto M. Mining telecommunications databases: an approach to support the business management. NOMS 98 1998 IEEE Network Operations and Management Symposium; 1998. doi:10.1109/noms.1998.654855im.2005.06.007.

172.Mancini, G., P. F. Ferrari, and E. Palagi. 2013. “Rapid Facial Mimicry in Geladas.” Scientific Reports 3: p. 1527. doi:10.1038/srep01527.

173.Manning A. databases for small business: essentials of database management, data analysis, and staff training for entrepreneurs and professionals. Chapter 10. New York, NY: Apress. 2015. doi:10.1007/978-1-4842-0277-7_10.

174.Mannino M, Hong SN, Choi IJ. Efficiency evaluation of data warehouse operations. Decis Supp Syst. 2008; 44(4): pp. 883–98. doi:10.1016/j.dss.2007.10.011.

175.Marks, M. A., M. J. Sabella, C. S. Burke, and S. J. Zaccaro. 2002. “The Impact of Cross-Training on Team Effectiveness.” The Journal of Applied Psychology 87 (1): pp. 3–13. doi:10.1037/0021-9010.87.1.3

176.Mauss, I. B., and M. D. Robinson. 2009. “Measures of Emotion: A Review.” Cognition and Emotion 23 (2): pp. 209–237. doi:10.1080/02699930802204677

177.McCarty, B., 2005. «Dynamics of a successful Team. What are the enablers and barriers to High Performing Successful Teams?» MSc Dissertation, Dublin City University.

178.Medykovskyj M., Pasyeka M., Pasyeka N. end Tyrchyn O. Scientific Research Life Cycle Performance Of Information Technology. XIIth International Scientific and Technical Conference «Computer Science & Information Technologies» (CSIT’2017), LVIV, Ukraine, 5–8 september 2017, pp. 425–428.

179.Microsoft Security Bulletin MS16-039 - Critical, 2017 (January 23). Security update for microsoft graphics component (3148522). Retrieved from https://docs.microsoft.com/en-us/security-updates/SecurityBulletins/2016/ms 16-039.

180.Milojević S. 2014 Principles of scientific research team formation and evolution. Proc. Natl Acad. Sci. USA 111, pp. 3984–3989. (doi:10.1073/pnas.1309723111).

181.Mockus A, Fielding RT, Herbsleb JD. 2002 Two case studies of open source software development: apache and mozilla. ACMTrans. Softw. Eng. Methodol. 11, pp. 309–346. (doi:10.1145/567793.567795).

182.Montague, E., and J. Xu. 2012. “Understanding Active and Passive Users: The Effects of an Active User Using Normal, Hard and Unreliable Technologies on User Assessment of Trust in Technology and co-User.” Applied Ergonomics 43 (4): pp. 702–712. doi:10.1016/j.apergo.2011.11.002

183.Montague, E., J. Xu, and E. Chiou. 2014. “Shared Experiences of Technology and Trust: An Experimental Study of Physiological Compliance between Active and Passive Users in Technology Mediated Collaborative Encounters.” IEEE Transactions on Human-Machine Systems 44 (5): pp. 614–624. doi:10.1109/THMS.2014.2325859.

184.Montjoye YA, Stopczynski A, Shmueli E, Pentland A, Lehmann S. 2014 The strength of the strongest ties in collaborative problem solving. Sci. Rep. 4, 5277. (doi:10.1038/srep05277).

185.Morgan, B., and S. K. D'Mello. 2016. “The Influence of Positive vs. negative Affect on Multitasking.” Acta Psychologica 170 (Supplement C): pp. 10–18. doi:10.1016/j.actpsy.2016.06.006.

186.Mtigwe, B. (2005). The entrepreneurial firm internationalization process in Southern African context: A comparative approach.

International Journal of Entrepreneurial Behaviour and Research, 11(5), pp. 358–377. doi:10.1108/13552550510615006.

187.Mucha PJ, Richardson T, Macon K, Porter MA, Onnela JP. 2010 Community structure in time-dependent, multiscale, and multiplex networks. Science 328, pp. 876–878. (doi:10.1126/science.1184819).

188.Müller, M. 2007. Dynamic Time Warping. Information retrieval for music and motion, pp. 69–84. New York: Springer.

189.N. B. Moe, T. Dingsøyr, and T. Dybå, «A teamwork model for understanding an agile team: A case study of a Scrum project,» Information and Software Technology, vol. 52, pp. 480-491, May. 2010.

190.Newman MEJ, Park J. 2003Why social networks are different from other types of networks. Phys. Rev. E. 68, 036122. (doi:10.1103/PhysRevE.68. 036122).

191.Newman MEJ. 2010 Networks: an introduction. Oxford, UK: Oxford University Press.

192.Ngo, F., & Jaishankar, K. (2017). Commemorating a decade in existence of the international journal of cyber criminology: a research agenda to advance the scholarship on cyber crime. International Journal of Cyber Criminology, 11, 1. doi:10.5281/zenodo.495762.

193.Niu, S., Mo, J., Zhang, Z., & Lv, Z. (2014). Overview of linux vulnerabilities. In 2nd International Conference on Soft Computing in Information Communication Technology. Atlantis Press. doi: 10.2991/scict-14.2014.55.

194.Nixon N. NongFu Spring Continues Growing With SAP HANA, SAP Business Trends; 2012 http://scn.sap.com/community/business-trends/blog/2012/08/07/nongfu-spring-grows-with-sap-hana.

195.Norwegian University of Science and Technology, Trondheim, Norway. Littlepage, G. E., Cowart, L., & Kerr, B. (1989). Relationships between group environment scales and group performance and cohesion. Small Group Research, 20(1), pp. 50–61. doi:10.1177/104649648902000104.

196.Palla G, Barabási AL, Vicsek T. 2007 Quantifying social group evolution. Nature 446, pp. 664–667. (doi:10.1038/nature05670).

197.Papp, D., Ma, Z., & Buttyan, L. (2015). Embedded systems security: Threats, vulnerabilities, and attack taxonomy. In Privacy, Security and Trust (PST), 2015 13th Annual Conference on pp. 145–152 doi: 10.1109/PST.2015.723296.

198.Pasyeka M., Parada R., Tyahun D. end Kosmirak V. The usage of mathematical tools Bezier curves in Flash game. XI-th International

Scientific and Technical Conference «Computer Science & Information Technologies» (CSIT'2014), LVIV, Ukraine, 18–22 November 2014. pp. 105–109.

199.Pasyeka M., Pasyeka N. end Yurchak I. Analysis of structural-logical models of learning process and their impact on improving the quality of education in the context of information technology. (MEMSTECH 2016) XII-th International conference. «Perspective technologies and methods in MEMS design», Polyana, Ukraine, 20–24 April 2016. pp. 140–143.

200.Pasyeka M., Pasyeka N., Danyliuk S. Methods for assessing the curriculum regarding information technology. Advances of Science Proceedings of articles the international scientific conference Czech Republic, Karlovy Vary – Ukraine, Kyiv, 28 September 2018. pp.1275-1283.

201.Pasyeka M., Pasyeka N., Yurchyshyn V., Kozak O. end Bandura V. Adaptive model evaluation test tasks of universities, as an element of improving the quality of education. XI-th International Scientific and Technical Conference. «Computer Science & Information Technologies» (CSIT'2014), LVIV, Ukraine,18–22 November 2014, pp. 122–126.

202.Pasyeka M.S. Development of Data Warehouse Structure Model for Educational Process Management. Software Engineering. Vol. 6, No. 1, 2018, pp. 1–6. doi: 10.11648/j.se.20180601.11.

203.Pasyeka M.S. Structure of the information science for intellectual analysis. «Proceedings of XXX Internationalscientific conference — Scientific development prospects». Morrisville, Lulu Press., 2018. pp. 23–26.

204.Pasyeka N., Mykhailyshyn H. and Pasyeka M. «Development Algorithmic Model for Optimization of Distributed Fault-Tolerant Web-Systems» IEEE International Scientific-Practical Conference «Problems of Infocommunications. Science and Technology» Kharkiv, 9–12 October, 2018, pp. 663-669.

205.Pasyeka N.M. end Pasyeka M.S. Construction of multidimensional data warehouse for processing students' knowledge evaluation in universities. 13–Міжнародна науково-техніч-на конференція (TCSET'2016), Національний університет «Львівська Політехніка», Львів–Славське, 23-26 лютого 2016, С. 822–824.

206.Pasyeka M. end Yurchak I. Designing the conceptual model of the management of higher. Вісник НУ «ЛП» - комп'ютерні системи проектування теорія і практика», м. Львів, № 808, 2014, С. 21–26.

207.Pasyeka M. end Yurchak I. Проектування концептуальної моделі управління вищим навчальним закладом основі інформаційних і інтерперативних Web-технологій. Матеріали XII Міжнародної науково-технічної конференції «Досвід розробки та застосування приладотехнологічних САПР в мікроелектроніці» «CAD in Machinery Design. Implementation and Educational Issues» (CADMD'2014), Lviv, Ukraine, 10–11 October 2014. pp. 34–45.

208.Pentland A. 2012 The new science of building great teams. Harv. Bus. Rev. 90, pp. 60–69.

209.Ployhart, R. E., J. A. Weekley, and K. Baughman. 2006. "The Structure and Function of Human Capital Emergence: A Multilevel Examination of the Attraction-Selection-Attrition Model." Academy of Management Journal 49 (4): pp. 661–677. doi:10.5465/AMJ.2006.22083023.

210.Poeplau, S., Fratantonio, Y., Bianchi, A., Kruegel, C., & Vigna, G. (2014). Execute this! analyzing unsafe and malicious dynamic code loading in android applications. In Network and Distributed System Security Symposium (NDSS) (Vol. 14, pp. 23–26. doi: 10.14722/ndss.2014.23328.

211.R Core Team. 2017. R: A Language and Environment for Statistical Computing. Vienna, Austria: R Foundation for Statistical Computing. http://www.R-project.org.

212.R. Lingard and S. Barkataki. «Teaching teamwork in engineering and computer science». In Frontiers in Education Conference. IEEE, Rapid City, SD, F1C pp. 1-5, 2011.

213.Ram, L. V. (2003, May 10). Guidelines on teambuilding. New Straits Times-Management Times. Rosen, C. C. H. (2005). The influence of intra team relationships on the systems development process: A theoretical framework of intra-group dynamics. In Proceedings of the 17th Workshop of the Psychology of Programming Interest Group pp. 30-42. Sussex, UK: PPIG.

214.Rhodes, N., K. Pivik, and M. Sutton. 2015. "Risky Driving among Young Male Drivers: The Effects of Mood and Passengers." Transportation Research Part F-Traffic Psychology and Behaviour 28: pp. 65–76. doi:10.1016/j.trf.2014.11.

215.Rising L, Janoff NS. 2000 The Scrum software development process for small teams. IEEE Softw. 17, pp. 26–32. (doi:10.1109/52.854065).

216.Rosen, C.C.H., «The Influence of Intra Team relationships on the systems Development Process: A theoretical Framework of Intra-Group Dynamics.», 17th Workshop of the Psychology off Programming Interest Group, Sussex University, 2005.

217.Roshna R, Punitha S, Sowbhagya M. Business intelligence solutions in retail. Int Proc Comput Sci Inf Technol. 2014; p. 59: 26.

218.S Basri and O' Connor, RV «Understanding the Perception of Very Small Software Companies towards the Adoption of Process Standards», Systems, Software and Services Process Improvement, Communications in Computer and Information Science, Volume 99, pp. 153-164, , Springer-Verlag, 2010.

219.S. Fincher, M. Petre, and M. Clark. Computer Science Project Work: Principles and Pragmatics. Springer Science & Business Media, London, UK, 2001.

220.S. J. Zaccaro, A. L. Rittman, M. A. Marks, «Team Leadership», The Leadership Quarterly vol. 12(4), pp. 451-483, 2001.

221.Salim N, Ibrahim R. Quality-based framework for requirement analysis in data warehouse. 2014 International Conference of Advanced Informatics: Concept, Theory and Application (ICAICTA); 2014. doi:10.1109/icaicta.2014.7005932.

222.Samaila, M. G., Neto, M., Fernandes, D. A. B., Freire, M. M., & Inácio, P. R. M. (2017). Security Challenges of the Internet of Things. Pp. 53–82, In J. Batalla, G. Mastorakis, C. Mavromoustakis, & E. Pallis (Eds.), Beyond the Internet of Things. Internet of Things (Technology, Communications and Computing) Cham: Springer. doi:10.1007/978-3-319-50758-3_3.

223.Scarnati, J. T. «On becoming a team player», Team Performance Management, Vol. 7, Issue.1/2, pp. 5 – 10, 2001.

224.Scherer, K. R. 2005. "What Are Emotions? and How Can They Be Measured?." Social Science Information 44 (4): pp. 695–729. doi:10.1177/05390184050582.

225.Scholtes I, Mavrodiev P, Schweitzer F. In press. From Aristotle to Ringelmann: a large-scale analysis of team productivity and coordination in Open Source Software projects. Empir. Softw. Eng. doi:10.1007/s10664-015-9406-4).

226.Schuff D, Corral K, Turetken O. Comparing the understandability of alternative data warehouse schemas: an empirical study. Decis Support Syst. 2011;52(1): pp. 9–20. doi:10.1016/j.dss.2011.04.003.

227.Sen A, Ramamurthy K, Sinha AP. A model of data warehousing process maturity. IEEE Trans Software Eng. 2012;38(2): pp. 336–53. doi:10.1109/TSE.2011.2.

228.Serrano M, Calero C, Trujillo J, Luján-Mora S, Piattini M. Empirical validation of metrics for conceptual models of data warehouses. International Conference on Advanced Information Systems Engineering pp. 506–20. Springer Berlin Heidelberg; 2004, Jun.

229.Serrano M, Trujillo J, Calero C, Piattini M. Metrics for data warehouse conceptual models understandability. Inf Software Technol. 2007;49(8):pp. 851–70. doi:10.1016/j.infsof.2006.09.008.

230.Sharma, C., & Jain, S. C. (2014, August). Analysis and classification of SQL injection vulnerabilities and attacks on web applications. International Conference on Advances in Engineering and Technology Research (ICAETR), 2014 pp. 1–6. IEEE. doi: 10.1109/ICAETR.2014.7012815.

231.Sheketa V.I., Vovk R.B., Pikh V.Y., Romanyshyn Y.L., Pasyeka M.S. Intelligent decisions support by oil&gas wells drilling. Міжнародна наукова конференція «Інтелектуальні системи прийняття рішень і проблеми обчислювального інтелекту» – ISDMCI'2019. с. Залізний Порт, Херсон, 21-25 травня 2019. С. 210-211.

232.Singh R, Singh K. A descriptive classification of causes of data quality problems in data warehousing. Int J Comput Sci Issue. 2010; 7(3): pp. 41–50.

233.Singh, S. K., «Role of leadership in knowledge management: A study», Journal of Knowledge Management, Vol.12, Issue. 4, pp. 3–15, 2008.

234.Sochor, T., Zuzcak, M., & Bujok, P. (2016). Analysis of attackers against windows emulating honeypots in various types of networks and regions. In Ubiquitous and Future Networks (ICUFN), 2016 Eighth International Conference on pp. 863–868. IEEE. doi: 10.1109/ICUFN.2016.7537159.

235.Song, C., Lee, B., Lu, K., Harris, W., Kim, T., & Lee, W. (2016). Enforcing kernel security invariants with data flow integrity. In Network and Distributed System Security Symposium (NDSS). doi:10.1109/INFOCOM.2015.7218424.

236.Sparrowe RT, Liden RC, Wayne SJ, KraimerML. 2001 Social networks and the performance of individuals and groups. Acad. Manage. J. 44, pp. 316–325. (doi:10.2307/3069458).

237.Storozh Y., Lyutak I., Storozh B., Vasylyk O., Yatsyshyn M. end Pasyeka M. Computer aided analysis of ball burnishing process. Международный научный журнал «Управляющие системы и машины», м. Київ, №5(259), 2015, С. 61–65.

238.Su Y, Peng J, Jin Z. Modeling information quality risk for data mining in data warehouses. Hum Ecol Risk Assess: Int J. 2009; 15(2): pp. 332–50. doi:10.1080/10807030902761411.

239.Sy, T., S. Côté, and R. Saavedra. 2005. "The Contagious Leader: Impact of the Leader's Mood on the Mood of Group Members, Group Affective Tone, and Group Processes." Journal of Applied Psychology 90 (2): pp. 295–305. doi:10.1037/0021-9010.90.2.295.

240.T. B. Hilburn and W. S. Humphrey. « Teaching Teamwork». IEEE Software 19, pp. 72–77, 2002.

241.T. Dingsøyr and Y. Lindsjørn, «Team performance in agile development teams: findings from 18 focus groups,» International Conference on Agile Software Development, Springer Berlin Heidelberg, June. 2013, pp. 46-60.

242.Upguard, 2017 (January 23). Top 20 critical windows server 2008 vulnerabilities and remediation tips. Retrieved from https://www.upguard.com/articles/top-20-critical-windows-server-2008-vulnerabilities-and-remediation-tips.

243.V. Lalsing, S. Kishnah, and P. Sameerchand, «People factors in agile software development and project management,» International Journal of Software Engineering & Applications, vol. 3, pp. 117, Jan.2012.

244.Valacich, J. S., A. R. Dennis, and J. F. Nunamaker. 1992. "Group Size and Anonymity Effects on Computer-Mediated Idea Generation." Small Group Research 23 (1): pp. 49–73. doi:10.1177/1046496492231004.

245.Valtanen, A., & Sihvonen, H. M. (2008). Employees' motivation for SPI: Case study in a small Finnish software company. In Proceedings of the 15th European Conference on Software Process Improvement pp. 152-163. Berlin, Germany: Springer-Verlag.

246.Van Mierlo, H., J. K. Vermunt, and C. G. Rutte. 2009. "Composing Group-Level Constructs from Individual-Level Survey Data." Organizational Research Methods 12 (2): pp. 368–392. doi:10.1177/1094428107309322.

247.Vargas, R. J. G., Huerta, R. G., Anaya, E. A., & Hernandez, A. F. M. (2012). Security controls for Android. In Computational Aspects of Social Networks (CASoN), 2012 Fourth International Conference on pp. 212–216. IEEE. doi: 10.1109/CASoN.2012.6412404.

248.Vespignani A. 2012 Modelling dynamical processes in complex socio-technical systems. Nat. Phys. 8, pp. 32–39. doi:10.1038/nphys2160.

249.Volpe, C. E., J. A. Cannon-Bowers, E. Salas, and P. E. Spector. 1996. “The Impact of Cross-Training on Team Functioning: An Empirical Investigation.” Human Factors: The Journal of the Human Factors and Ergonomics Society 38 (1): pp. 87–100. doi:10.1518/001872096778940741.

250.Wall, D. S. (2012). The devil drives a Lada: The social construction of hackers as cybercriminals. Constructing Crime pp. pp. 4–18. Palgrave Macmillan, London. doi:10.1057/9780230392083_2.

251.Wang, T., Jang, Y., Chen, Y., Chung, S. P., Lau, B., & Lee, W. (2014). On the feasibility of large-scale infections of iOS devices. In USENIX Security Symposium pp. 79–93. 10.1177/1753193412471103.

252.Watts DJ, Dodds PS, Newman MEJ. 2002 Identity and search in social networks. Science 296, pp. 1302–1305. doi:10.1126/science.1070120.

253.West, B. T., K. B. Welch, and A. T. Galecki. 2014. Linear Mixed Models: A Practical Guide Using Statistical Software. 2nd ed. Boca Raton, FL: CRC Press.

254.Westerman P. data warehousing: Using the wal-mart model. San Francisco, CA: Morgan Kaufmann; 2001.

255.White, I. R., P. Royston, and A. M. Wood. 2011. “Multiple Imputation Using Chained Equations: issues and Guidance for Practice.” Statistics in Medicine 30 (4): pp. 377–399. doi:10.1002/sim.4067.

256.Wiles, J. A., and T. B. Cornwell. 1991. “A Review of Methods Utilized in Measuring Affect, Feelings, and Emotion in Advertising.” Current Issues and Research in Advertising 13 (1–2): pp. 241–275. doi:10.1080/01633392.1991.10504968.

257.Winter R, Klesse M. Manage business metadata and ensure information quality. Busi Intell J. 2009 Third;14: pp. 31–39.

258.X. Kong, L. Liu, and D. Lowe, «Modeling an agile web maintenance process using system dynamics,» In 11th ANZSYS/Managing the Complex V conference, ISCE Publishing, Christchurch, NZ. Dec. 2005.

259.Xing, L., Bai, X., Li, T., Wang, X., Chen, K., Liao, X., & Han, X. (2015). Cracking app isolation on apple: Unauthorized cross-app resource access on mac os. In Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security pp. 31–43. ACM. doi: 10.1145/2810103.2813609.

260.Xolocotzin Eligio, U., S. E. Ainsworth, and C. K. Crook. 2012. “Emotion Understanding and Performance during Computer-Supported Collaboration.” Computers in Human Behavior 28 (6): pp. 2046–2054. doi:10.1016/j.chb.2012.06.001.

261.Xu, J., and E. Montague. 2012. “Psychophysiology of the Passive User: Exploring the Effect of Technological Conditions and Personality Traits.” International Journal of Industrial Ergonomics 42 (5): pp. 505–512. doi:10.1016/j.ergon.2012.07.007.

262.Xu, J., and E. Montague. 2015. “Affect and Trust in Technology in Teams: The Effect of Incidental Affect and Integral Affect.” Proceedings of the Human Factors and Ergonomics Society Annual Meeting 59 (1): pp. 205–209.

263.Xu, J., K. Le, A. Deitermann, and E. Montague. 2014. “How Different Types of Users Develop Trust in Technology: A Qualitative Analysis of the Antecedents of Active and Passive User Trust in a Shared Technology.” Applied Ergonomics 45 (6): pp. 1495–1503. doi:10.1016/j.apergo.2014.04.012.

264.Y. Ramírez and D. Nembhard, «Measuring knowledge worker productivity: A taxonomy,» Journal of Intellectual Capital, vol. 5, no. 4, Dec. 2004, pp. 602–628.

265.Yatsyshyn M., Yurchushun V., Storog Y. end M. Pasyeka Information-Categorical Formalization of Processes and Objects of Oil and Gas Object Domain. International Journal of Oil, Gas and Coal Engineering 2017; Vol. 5, No. 6, 2017, pp. 184-188. doi: 10.11648/j.ogce.20170506.19.

266.Zavou, A., Athanasopoulos, E., Portokalidis, G., & Keromytis, A. D. (2012). Exploiting split browsers for efficiently protecting user data. In Proceedings of the 2012 ACM Workshop on Cloud computing security workshop pp. 37–42. ACM. doi: 10.1145/2381913.2381921.

267.Zhang, M., & Yin, H. (2014). AppSealer: automatic generation of vulnerability-specific patches for preventing component hijacking attacks in android applications. In Network and Distributed System Security Symposium (NDSS). doi: 10.14722/ndss.2014.23255.

268.Басюк Т. М., Жежнич П. І. Методи та засоби мультимедійних інформаційних систем. Навчальний посібник. Львів : Видавництво Львівської політехніки, 2015. 428 с.

269.Білас О. Є. Якість програмного забезпечення та тестування. Навчальний посібник. Львів: Видавництво Львівської політехніки, 2011. 216 с

270.Бодянський Є. В. та ін. Аналіз та обробка потоків даних засобами обчислювального інтелекту. Монографія / Є. В. Бодянський, Д. Д. Пелешко, О. А. Винокурова, С. В. Машталір, Ю. С. Іванов. Львів : Видавництво Львівської політехніки, 2016. 236 с.

271.Буров Є. В. Концептуальне моделювання інтелектуальних програмних систем. Монографія. Львів: Видавництво Львівської політехніки, 2012. 432 с.

272.Верес О. М. Технології підтримання прийняття рішень. Навчальний посібник. Друге видання. Львів: Видавництво Львівської політехніки, 2013. 252 с.

273.Грень Я.В. Програмування систем реального часу. Навчальний посібник. Львів: Видавництво Львівської політехніки, 2011. 324 с.

274.Дурняк Б., Пасєка М.С., Пасєка Н. end Ерстенюк О. Проектування та використання сховищ даних для опрацювання результатів оцінювання знань студентів. Науковий вісник НЛТУ України: Збірник науково-технічних праць. – Львів: РВВ НЛТУ України. – Вип. 25.9. 2015. С. 365–373.

275.Дурняк Б. В., Михайлишин Г. Й, Пасєка Н. М., Пасєка М. С., Майба Т. М. Інформаційна технологія активізації навчання. Монографія. [Української академії друкарства]. 2019. – 184 с.

276.Дурняк Б. В., Пасєка М. С., Майба Т. М. Управління запитами в системах документообігу. Монографія. [Української академії друкарства]. 2016. – 194 с.

277.Жежнич П. І. Консолідовані інформаційні ресурси баз даних та знань. Навчальний посібник. Львів: Видавництво Львівської політехніки, 2010. (Серія “Консолідована інформація”. Випуск 7). 212 с.

278.Заяць В. М. Методи і засоби комп’ютерних інформаційних технологій. Навчальний посібник. Львів: Видавництво Львівської політехніки, 2013. 144 с.

279.Зорін В., Бандура В., Храбатин Р., Пасєка М., Пасєка Н. Удосконалення методів оцінки набуття знань студентів на основі дистанційного навчання. Вісник Національного університету «Львівська політехніка» серія «Інформатизація вищого навчального закладу», м. Львів, 2016. С. 131–137.

280.Зорін В., Бандура В., Юрчишин В., Пасєка М. Системний аналіз факторних впливів на групову динаміку при розробці програмного забезпечення. ISDMCI’2015 International Conference Intellectual Systems for Decision Making and Problems of Computational Intelligence: Conference Proceedings. Kherson, KNTU, 2015. pp. 70–72.

281.Коротєєва Т. О. Алгоритми та структури даних. Навчальний посібник / Т. О. Коротєєва. Львів : Видавництво Львівської політехніки, 2014. – 280 с.

282.Кравець П. Об'єктно-орієнтоване програмування. Навчальний посібник. Львів: Видавництво Львівської політехніки, 2012. 624 с.

283.Кравець Р. Б. та ін. Інформаційні технології організації бізнесу. Навчальний посібник / Р. Б. Кравець, Ю. О. Сєров, О. В. Марковець. Львів: Видавництво Львівської політехніки, 2013. 228 с.

284.Кунанець Н. Е., Пасічник В. В. Вступ до фаху "Консолідована інформація". Навчальний посібник. Друге видання. Львів: Видавництво Львівської політехніки, 2013. 196 с.

285.Лагун А. Е. Криптографічні системи та протоколи. Навчальний посібник. Львів: Видавництво Львівської політехніки, 2013. 96 с.

286.Литвин В. В. Бази знань інтелектуальних систем підтримки прийняття рішень. Монографія. Львів: Видавництво Львівської політехніки, 2011. 240 с.

287.Литвин В. В. Технології менеджменту знань. Навчальний посібник / За заг. ред. В. В. Пасічника. Друге видання. Львів: Видавництво Львівської політехніки, 2013. 260 с.

288.Мацибурка П. Т., Пасєка М. С. Архітектурні особливості розгортання ВЕБ–сервісу. «Науково-технічна конференція професорсько-викладацького складу, наукових працівників і аспірантів» «Українська академія друкарства» м. Львів, 14-17 лютого 2017. С. 126.

289.Мацибурка П. Т., Пасєка М. С. Дослідження методів та інструментів побудови високонавантажених розподілених систем для Web-скрапінгу. III Всеукраїнська науково-практична конференція молодих учених і студентів «Інформаційні технології в освіті техніці та промисловості», м. Івано-Франківський, 10-13 жовтня 2017. С. 30–32.

290.Мацибурка П. Т., Пасєка М. С. Особливості розгортання Web-сервісу для дистанційної освіти. 5-ий Всеукраїнський науково-практичний семінар «Сучасні технології в дистанційній освіті», 25–28 травня, м. Івано–Франківськ, 2017. С. 81–83.

291.Медиковський М. О. та ін. Інтелектуальні компоненти інтегрованих автоматизованих систем управління. Монографія / М. О. Медиковський, Р. О. Ткаченко, І. Г. Цмоць, Ю. В. Цимбал, А. В. Дорошенко, О. В. Скорохода. Львів : Видавництво Львівської політехніки, 2015. 280 с.

292.Мельник Р. А., Тушницький Р. Б. Програмування інтернет-застосувань. Навчальний посібник. Львів : Видавництво Львівської політехніки, 2013. 256 с.

293.Мина Ж. В. Аналітико-синтетичне опрацювання інформації. Навчальний посібник. Львів: Видавництво Львівської політехніки, 2016. 196 с .

294.Мина Ж. В., Думанський Н. О. Документаційне забезпечення діяльності установи. Навчальний посібник. Львів : Видавництво Львівської політехніки, 2015. 158 с.

295.Міюшкович Ю. Г. та ін. Проектування систем засобами AllFusion Modeling Suite. Навчальний посібник / Ю. Г. Міюшкович, Р. С. Марцишин, Л. С. Сікора. Львів : Видавництво Львівської політехніки, 2014. 156 с.

296.Навчальний посібник / За заг. ред. В. В. Пасічника. Львів: Видавництво Львівської політехніки, 2010. (Серія "Консолідована інформація". Випуск 4). 248 с.

297.Павлиш В. А., Гліненко Л. К. Основи інформаційних технологій і систем. Навчальний посібник. Львів: Видавництво Львівської політехніки, 2013. 500 с.

298.Пасєка М. С., Пасєка Н. М., Бестильний М. Я., Шекета В. І. Аналіз використання високоефективної реалізації функцій хешування sha-512 для розробки програмних систем. Кібербезпека: освіта, наука, техніка. Київський університет імені Бориса Грінченка № 3 (3) 2019 С. 112-121.

299.Пасєка М., Пасєка Н. and Ерстенюк О. Застосування моделей структурно-логічних схем у навчальному процесі. Науковий вісник НЛТУ України: Збірник науково-технічних праць. – Львів: РВВ НЛТУ України. – Вип. 26.4. 2016. С. 409–413.

300.Пасєка М., Пасєка Н., Бандура В., Храбатин Р. Методи оцінювання набуття компетенцій студентів дистанційної форми навчання. Наукові записки [Української академії друкарства]. № 2 (53) – 2016 С. 120–126.

301.Пасєка М., Пасєка Н., Корнута Ю. Аналіз концепцій побудови відмовостійких розподілених Web–систем. Міжнародну науково-практичну конференцію молодих вчених та студентів «Інформаційні технології, економіка та право: стан та перспективи розвитку» (ІТЕП-2018) м. Чернівці, 19–20 квітня 2018. С. 58–59.

302.Пасєка М., Турчин О. Теоретичне обґрунтування показників життєвого циклу технологій. Науковий вісник НЛТУ України: Збір-

ник науково-технічних праць. – Львів: РВВ НЛТУ України. – Вип. 26.3. 2016. С. 334–341.

303.Пасєка М. С. Використання інформаційних технологій для професійної підготовки фахівців у галузі педагогіки. Науковий вісник НЛТУ України: Збірник науково-технічних праць. – Львів: РВВ НЛТУ України.– Вип. 25.5. 2015 С. 357-362.

304.Пасєка М. С. Використання хмарних технологій для розробки програмних додатків. Поліграфія і видавнича справа / Printing and publishing № 1 (73) / 2017 – м.Львів, С. 70–79.

305.Пасєка М. С. Опрацювання даних адаптивного навчання і тестування студентів вищого навчального закладу. Науковий вісник НЛТУ України: Збірник науково-технічних праць. – Львів: РВВ НЛТУ України. – Вип. 25.4. 2015 С. 400–407.

306.Пасєка М. С. Особливості групової динаміки в інженерії програмного забезпечення. Науковий вісник НЛТУ України: Збірник науково-технічних праць. – Львів: РВВ НЛТУ України. – Вип. 25.3. 2015. С. 391–397.

307.Пасєка М. С. Системний аналіз хмарної технології для розробки програмних додатків. Вчені записки Таврійського національного університету імені В. І. Вернадського м. Київ Том 29 (68) № 6 2018. С. 206 – 211.

308.Пасєка М. С. Хмарний сервіс для управління розробкою програмних додатків на базі сервера Redmine. Кваліологія книги, м. Львів, № 1 (31) 2017. С. 73–81.

309.Пасєка М., Пасєка Н. end Ерстенюк О. Структуризація методів розв’язання математичних і прикладних задач та їх інформаційна сутність. Науковий вісник НЛТУ України: Збірник науково-технічних праць. – Львів: РВВ НЛТУ України – Вип. 26.4.– 2016. С. 409 – 413.

310.Пасєка Н., Пасєка М., Храбатин Р., Юрчишин В., Бандура В. Методи оцінки якості набуття компетенцій студентами дистанційної форми навчання. IV–всеукраїнський науково-практичний семінар «Сучасні інформаційні технології в дистанційній освіті», м. Івано-Франківськ, 21–23 вересня 2015. С. 22–25.

311.Пасєка Н. М., Пасєка М. С. Методи оцінки набуття компетенцій студентами дистанційної форми навчання. IV–й Всеукраїнський науково-практичний семінар «Сучасні інформаційні технології в дистанційній освіті» (MITDE-2015), Івано-Франківський національний технічний університет нафти і газу, 2015. С. 22–24.

312.Пелешко Д. Д., Теслюк В. М. Об’єктні технології С++11. Навчальний посібник. Львів: Видавництво Львівської політехніки, 2013. 360 с.

313.Пелещишин А. М. Інтернет-технології опрацювання консолідованих інформаційних ресурсів. Навчальний посібник. Львів: Видавництво Львівської політехніки, 2015. 257 с.

314.Пелещишин А. М. та ін. Процеси управління інтерактивними соціальними комунікаціями в умовах розвитку інформаційного суспільства. Монографія / А. М. Пелещишин, Ю. О. Сєров, О. Л. Березко, О. П. Пелещишин, О. Ю. Тимовчак-Максимець, О. В. Марковець. Львів: Видавництво Львівської політехніки, 2012. 368 с.

315.Ромака В. А. та ін. Системи менеджменту інформаційної безпеки. Навчальний посібник / В. А. Ромака, В. Б. Дудикевич, Ю. Р. Гарасим, П. І. Гаранюк, І. О. Козлюк. Львів: Видавництво Львівської політехніки, 2012. 232 с.

316.Савич Ю., Вовк Р., Пасєка М. Аналіз методів криптографічного захисту інформації. VIII–Всеукраїнська науково практична конференція студентів, аспірантів, молодих вчених «Інформаційні технології безпека та зв’язок» м. Дніпро, 24 лисопада 2016. С. 6–7.

317.Сенів М. М., Яковина В. С. Безпека програм та даних. Навчальний посібник. Львів : Видавництво Львівської політехніки, 2015. 256 с.

318.Ткаченко Р. О. та ін. Засоби штучного інтелекту. Навчальний посібник / Р. О. Ткаченко, Н. О. Кустра, О. М. Павлюк, У. В. Поліщук. Львів: Видавництво Львівської політехніки, 2014. 204 с.

319.Турчин О. Б., Пасєка М. С. Моделі, методи та алгоритми побудови розподілених відмовостійких Web-систем. III Всеукраїнська науково–практична конференція молодих учених і студентів «Інформаційні технології в освіті, техніці та промисловості», м. Івано-Франківський, 10–13 жовтня 2017. С. 96–97.

320.Турчин О. Б., Пасєка М. С. Перспективи розвитку розпаралелення обробки даних. «Науково-технічна конференція професорсько-викладацького складу, наукових працівників і аспірантів» «Українська академія друкарства» м. Львів, 14–17 лютого 2017. С. 127.

321.Федорчук Є. Н. Програмування систем штучного інтелекту. Експертні системи. Навчальний посібник. Львів: Видавництво Львівської політехніки, 2012. 168 с.

322.Химиця Н. О., Морушко О. О. Ділова комунікація Навчальний посібник. Львів: Видавництво Львівської політехніки, 2016. 208 с.

323. Храбатин Р., Бандура В., Пасєка М., Храбатин О., Лещишин Ю. Впровадження системи дистанційного навчання у ВНЗ для іноземних студентів: проблеми і перспективи. ІІІ–Міжнародна науково-практична конференція «Актуальні питання організації навчання іноземних студентів в Україні» м. Тернопіль, Тернопільський національний технічний університет ім. І. Полюя, 18–20 травня 2016, С. 188–190.

324. Шаховська Н. Б. Програмне та алгоритмічне забезпечення сховищ та просторів даних. Монографія. Львів: Видавництво Львівської політехніки, 2011. 196 с.

325. Шаховська Н. Б., Пасічник В. В. Сховища та простори даних. Монографія. Львів: Видавництво Львівської політехніки, 2009. 244 с.

326. Шпак З. Я. Програмування мовою С. Навчальний посібник. Друге видання, доповнене. Львів: Видавництво Львівської політехніки, 2011. 436 с.

327. Юрчишин В., Пасєка М., Бандура В. Концептуальна модель підвищення тенцій студентів нафтогазової промисловості в умовах дистанційного навчання на основі інформаційних і інтерперативних Web–технологій. ІІІ–й Всеукраїнський науково-практичний семінар «Сучасні інформаційні технології в дистанційній освіті» (MITDE–2014), Івано-Франківський національний технічний університет нафти і газу, 2014. С. 50–55.

328. Юрчишин В., Пасєка М., Храбатин Р., Бандура В. Інноваційно інформаційно-комунікаційні технології донесення навчального матеріалу у дистанційній освіті. 5–ий Всеукраїнський науково-практичний семінар «Сучасні технології в дистанційній освіті», 25–28 травня 2017, м. Івано-франківськ, С. 37–39.

329. Яковина В. С. та ін. Моделі, методи та засоби аналізу надійності програмних систем. Монографія / В. С. Яковина, Д. В. Федасюк, М. М. Сенів, О. О. Нитребич. Львів : Видавництво Львівської політехніки, 2015. 220 с.

330. Яцишин М., Піндак Р., Юрич А., Пасєка М. Концептуально-інформаційне моделювання аварійних ситуацій при бурінні нафтових і газових свердловин. Збірник наукових праць «Національна академія наук України» Моделювання та інформаційні технології. м.Київ, № 78, 2017. С. 121–132.

Наукове видання

Микола Степанович ПАСЄКА,
Надія Мирославівна Пасєка,
Юлія Любомирівна РОМАНИШИН
Василь Іванович ШЕКЕТА

ГРУПОВА ДИНАМІКА ЕФЕКТИВНИХ КОМАНД РОЗРОБНИКІВ

МОНОГРАФІЯ

Видавництво Івано-Франківського національного технічного університету нафти і газу

вул. Карпатська, 15, м. Івано-Франківськ, 76019, Україна
тел. (0342) 547266, факс (0342) 547139,
http://nung.edu.ua, e-mail: admin@nung.edu.ua

Свідоцтво про внесення до Державного реєстру видавців
ІФ № 18 від 12.03.2002 р.

Підписано до друку _10.01.2022 р.__ Формат _60×84 1/16_. Друк офсетний.

Ум. друк. арк. 17,55 . Тираж _300_ прим. Замовл. № _____